Dibris

University of Genoa

Department of Informatics, Bioengineering, Robotics and Systems Engineering (DIBRIS)

PhD Program in Bioengineering and Robotics, XXX Cycle

Curriculum in Robotics and Autonomous Systems

Maintaining Structured Experiences for Robots via Human Demonstrations

an Architecture to Convey Long-Term Robot's Beliefs

Author
Luca Buoncompagni

Supervisor
Fulvio Mastrogiovanni

Head of the Program
Giorgio Cannata

Thesis Jury
Luca Iocchi
Sapienza University of Rome

Alessandro Saffiotti
Örebro University

December 2018

Thesis submitted for the degree of Doctor of Philosophy

Luca Buoncompagni: *Maintaining Structured Experiences for Robots via Human Demonstrations,* an Architecture to Convey Long-Term Robot's Beliefs.

A thesis submitted for the degree of Doctor of Philosophy of the PhD Program in Bioengineering and Robotics, XXX Cycle. Curriculum in Robotics and Autonomous Systems. Department of Informatics, Bioengineering, Robotics and Systems Engineering (DIBRIS). University of Genoa.

AUTHOR:
Luca Buoncompagni luca.buoncompagni@edu.unige.it
buon_luca@yahoo.com

SUPERVISOR:
Fulvio Mastrogiovanni fulvio.mastrogiovanni@unige.it

A MIO PADRE, MIA MADRE E MIO FRATELLO,
LORO NON SMETTERANNO MAI DI INSEGNARMI.

TO MY FATHER, MOTHER AND BROTHER,
THEY WOULD NEVER STOP TEACHING ME.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Genoa (IT)
Dicember 2018
Luca Buoncompagni

Abstract

This PhD thesis presents an architecture for structuring experiences, learned through demonstrations, in a robot memory. To test our architecture, we consider a specific application where a robot learns how objects are spatially arranged in a tabletop scenario.

We use this application as a mean to present a few software development guidelines for building architecture for similar scenarios, where a robot is able to interact with a user through a qualitative shared knowledge stored in its memory. In particular, the thesis proposes a novel technique for deploying ontologies in a robotic architecture based on semantic interfaces. To better support those interfaces, it also presents general-purpose tools especially designed for an iterative development process, which is suitable for Human-Robot Interaction scenarios.

We considered ourselves at the beginning of the first iteration of the design process, and our objective was to build a flexible architecture through which evaluate different heuristic during further development iterations.

Our architecture is based on a novel algorithm performing a one-shot structured learning based on logic formalism. We used a fuzzy ontology for dealing with uncertain environments, and we integrated the algorithm in the architecture based on a specific semantic interface.

The algorithm is used for building experience graphs encoded in the robot's memory that can be used for recognising and associating situations after a knowledge bootstrapping phase. During this phase, a user is supposed to teach and supervise the beliefs of the robot through multimodal, not physical, interactions. We used the algorithm to implement a cognitive like memory involving the encoding, storing, retrieving, consolidating, and forgetting behaviours, and we showed that our flexible design pattern could be used for building architectures where contextualised memories are managed with different purposes, i.e. they contains representation of the same experience encoded with different semantics.

The proposed architecture has the main purposes of generating and maintaining knowledge in memory, but it can be directly interfaced with perceiving and acting components if they provide, or require, symbolical knowledge. With the purposes of showing the type of data considered as inputs and outputs in our tests, this thesis also presents components to evaluate point clouds, engage dialogues, perform late data fusion and simulate the search of a target position. Nevertheless, our design pattern is not meant to be coupled only with those components, which indeed have a large room of improvement.

Preface

This thesis reports the research I made during my PhD at the EMARO-lab, part of the DIBRIS department at the University of Genoa, from November 2015 to December 2018. Together with my supervisor, Prof. Fulvio Mastrogiovanni, I investigated open issues for effective knowledge representation in human-robot interaction scenarios. We argued a lack of methodologies for building memories for robots, and we decided to investigate a framework concerning qualitative and symbolic knowledge representation. Together with Teseo s.r.l., we are also considering similar methodologies for representing knowledge for human activity recognition in smart environments.

During this period, I also worked as teaching assistant for the course of Software Architecture for Robotic held by Fulvio Mastrogiovanni for the master degree in Computer Science, and for the European Master in Advanced Robotics (EMARO$^+$). During this experience, I had the opportunity to propose a few projects and supervise several skilled students on those projects. Some of the results obtained by the students contributed to the overall work presented in this thesis as detailed in the following list. I also visited the Cognitive Robotic Systems laboratory at the Center for Applied Autonomous Sensor Systems (AASS) of Örebro University, in Sweden. Thanks to the group lead by Prof. Alessandro Saffiotti, I could certainly improve our approach for representing knowledge toward real applications.

This thesis introduces a novel technique for representing dynamic knowledge through a collection of papers, which are listed below. It contains, four short papers submitted to workshops at international conferences and two accepted conferences papers. Also, it presents three journal papers that we aim to summit for peer-review. All of them concern different components of the same framework, through which, we build the architectures presented in the last part of the thesis.

Along with the architecture for the particular scenario that we considered, the thesis highlights some guidelines for building an architecture in similar circumstances. Unfortunately, we were not able to define a general framework due to the complexity of human-robot interaction scenarios, but, in the papers proposed below, we motivate some development patterns and tools for implementing memories for robots.

Even if the thesis proposes examples and simple use cases, we considered the issues of a real experimental setup and, for some of them, we propose possible solutions based on practical implementations.

Indeed, the thesis addresses also some technical issues led from real experiments where mainstream software packages were deployed. For each paper, the thesis indicates our open repositories containing the implementation we developed. Although there is a constant and ongoing effort for improving our software, it is difficult to keep the repositories up to date and bug-free. Nevertheless, I am happy to be able to share the implementations related to the contents of this thesis.

Follow a list of the accepted publications that are chapters of this thesis. The chapters have a different layout depending on the published resources, but the content is unchanged. Among the chapters, the sections with duplicated contents are removed whenever possible and substituted with side notes.

[1] Luca Buoncompagni, Alessio Capitanelli, and Fulvio Mastrogiovanni. "A ROS Multi-Ontology References Services: OWL Reasoners and Application Prototyping Issues." In: *Proceedings of the 5th Italian Workshop on Artificial Intelligence and Robotics A Workshop of the XVII International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018)*. Trento, Italy: CEUR-WS, 2018.

[2] Luca Buoncompagni, Alessandro Carfì, and Fulvio Mastrogiovanni. "A Software Architecture for Multimodal Semantic Perception Fusion." In: *Proceedings of the 5th Italian Workshop on Artificial Intelligence and Robotics A Workshop of the XVII International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018)*. Trento, Italy: CEUR-WS, 2018.

[3] Luca Buoncompagni and Fulvio Mastrogiovanni. "An Open Framework to Develop and Validate Techniques for Speech Analysis." In: *Proceedings of the 3rd Italian Workshop on Artificial Intelligence and Robotics A workshop of the XV International Conference of the Italian Association for Artificial Intelligence (AI*IA 2016)*. Genova, Italy: CEUR-WS, pp. 15–20.

[4] Luca Buoncompagni and Fulvio Mastrogiovanni. "A Software Architecture for Object Perception and Semantic Representation." In: *Proceedings of the 2nd Italian Workshop on Artificial Intelligence and Robotics A workshop of the XIV International Conference of the Italian Association for Artificial Intelligence (AI*IA 2015)*. Ferrara, Italy: CEUR-WS, 2015, pp. 116–124.

[5] Luca Buoncompagni and Fulvio Mastrogiovanni. "Dialogue-Based Supervision and Explanation of Robot Spatial Beliefs: a Software Architecture Perspective." In: *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 2018, pp. 977–984.

[6] Luca Buoncompagni, Suman Ghosh, Mateus Moura, and Fulvio Mastrogiovanni. "A Scalable Architecture to Design Multimodal Interactions for Qualitative Robot Navigation." In: *International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018)*. Springer. Trento, Italy, 2018, pp. 96–109.

Moreover, the chapters related to papers that we are going to submit for publication are

[7] Luca Buoncompagni, Syed Yusha Kareem, and Fulvio Mastrogiovanni. *OWLOOP: an API to Describe Dynamic OWL Axioms in OOP Objects*.

[8] Luca Buoncompagni and Fulvio Mastrogiovanni. *One-Shot Structured Learning of Scene Categories Through Demonstrations*.

[9] Luca Buoncompagni, Fulvio Mastrogiovanni, and Alessandro Saffiotti. *One-Shot Structured Learning of Robot's Experience in Uncertain Environments*.

[10] Luca Buoncompagni, Carlotta Sartore, and Fulvio Mastrogiovanni. *A Generalised Architecture to Encode, Store, Consolidate, Retrieve, and Forget Structured Experience for Robot's Memories*.

Once again I want to thank all the coauthors, and I would like to point the reader to Section 1.4 for an overview of the parts and chapters of this thesis.

errare humanum est
perseverare scentiam iuvat

Acknowledgments

I want to express my sincere gratitude to many people supporting this thesis. Firstly my supervisor, Prof. Fulvio Mastrogiovanni, for the always prompt and wise advice during my PhD experience, related networking, and life's problems. I could not have imagined having a better advisor and mentor for my study.

Also, I want to thank Prof. Alessandro Saffiotti for have kindly hosted me at the University of Örebro for a short but intense internship, without which, I could not present results for real-world setups.

This thesis would not have been possible without the lovely support of Aiko Dinale, who helped me in improving and writing my research. She spares no effort for sustaining me.

All my thanks to labmates and colleagues, Alessandro Carfì, Alessio Capitanelli, Alice Palazzo, Andrea Nisticò, Barbara Bruno, Giovanna Naselli, Roberto Menicatti and Syed Yusha Kareem, for the stimulating discussions, for the restless time we were working together, and for all the fun we have had in the last four years.

Thanks also to all the students I supervised, especially Carlotta Sartore, Suman Ghosh and Mateus Moura, for helping me to review my work from new inspiring perspectives.

Last but not least, I would like to thank my family and old friends for their spiritual guidance that brings me here.

Contents

List of Figures

List of Tables

List of Algorithms and Listings

Part I

Memory for Human-Robot Interaction

This part gives an overview of the thesis. It introduces the motivations, objective, contribution and structure of this thesis.

1

Introduction

In Human-Robot Interaction (HRI) scenarios, a user exploits cognitive abilities, as autonomous mechanisms, for perceiving the environment and learning from experiences to act toward a goal and flexibly make a prediction [161]. In this context, the human memory is a complex structure containing the knowledge that a user requires to perform all those interwoven abilities, which are also undertaken while interacting with a robot. We suppose that a robot interacts well with a user if it can cope with such abilities up to a level that is required for a particular task. For achieving this behaviour, we focus on the problem of generating, using, and maintaining robots' memories represented with the proper level of detail they require for interacting with users toward a shared goal.

Regarding the usability of the robot through interaction, there is an increasing interest in encoding cognitive aspects, such as social factors for instance [44]. In particular, implementations of the theory of mind has been proposed for HRI, as presented in [135] for instance. In accordance with this theory, a human empathetically interacts taking into account also a projection of the beliefs of the people interacting with.

In a practical situation, *e.g.* where persons have a shared goal in an assembly task, it has been observed that their interaction become easier when they start to know each other. In [81], this effect has been identified as an interaction gap, that the agents should overcome to improve their collaboration. We consider the interaction gap to be dependent on the context, *i.e.* two persons might have a small gap while performing a task usually done together, but a big gap for unfamiliar tasks. In other words, the persons contextualise their knowledge and focus their attention only on some relevant aspects of the environment. Such knowledge is structured from a different perspective in each person's memory, which also includes a projection of the context from the point of view of the other persons performing the same task [161].

Indeed, while we interact with a person, we communicate with him or her also with the intent to understand the other's beliefs and correct our projection of their beliefs. Eventually, this process improves the prediction that a person makes about the behaviour of another,

and this reduces the gap. This suggested, that while interacting, a user naturally attempts to identify cognitive robot's beliefs, even if it is supposed not to have them. Therefore, not only the robot should memorise assessments about the environment for making itself able to achieve a task, but also the persons should be able to understand the robot's beliefs based on verbal, and not verbal communication as well. We argue that if the robot's memory is structured similarly to the beliefs that a person has about a situation, such a communication will be more efficient, and this could improve their interaction.

Typically, it is considered that a person stores in the memory something that he or she learned, which, in most of the cases, is explained to others persons for simplifying some aspects of the interaction. Nowadays, there are many methodologies for making a robot able to recognise a task based on data-driven learning, but those approaches are intrinsically different from one-shot learning behaviours of humans while interacting. However, the fact that we lack systems that can explain the knowledge they learned and why a recognition occurred, is a limitation that affects the interaction. Since models learned in a data-driven manner are not based on a representation familiar to a person, it could be almost impossible for a user to understand and supervise the knowledge the robot has in its memory. Moreover, there are several approaches to fusing multi interaction modalities to implement a more immersive experience for the user and improve the understandings of the robot. Nevertheless, those techniques are typically strongly dependent on the type of data perceived, as well as on the application requirements.

A typical approach used for designing software architectures for robots that involve several functionalities is to introduce a layer describing the data required for orchestrating the components through a common syntax, which is used by all the other components. In this way, the interface between the components can be flexibly designed; this approach has been adopted for instance in [18, 29, 134], where the Ontology Web Language (OWL) has been used as common syntax [108]. OWL is a W3C standard implementation of Description Logic (DL) formalism [11]. Such a symbolic formalism allow to structure knowledge in logic terms, and it provides with standard reasoners evaluating the semantics of a given domain for inferring new knowledge, as well as responding to queries with retrieving purposes. OWL has been successfully used in robotic and ambient intelligent scenarios for recognising models in a contextualised fashion [83], and it has been used for representing qualitative knowledge for performing commonsense reasoning as well. Remarkably, OWL allows simplifying the interaction since it generates representations that are familiar to persons and, even if DL does not assure that all reasoning processes are always easily understandable by users, its de-

scriptive representation has been a critical feature also for developing cognitive architectures [93], as well as planners [32].

Nevertheless, such symbolic representations lead a few issues that we will discuss later in the thesis and, for some of them, we will provide a possible solution. For some applications, the overhead for maintaining a complex representation of the robot's memory might limit its abilities, and for some simple tasks, it would be more efficient to use not shared data structures. On the other hand, the interaction could be improved if the robot behaves in order to align its beliefs with the ones of the user. Finding the correct levels of details in the memory to be maintained, is a matter of choosing a trade-off among the complexity of the robot's knowledge and the richness of its behaviour. Indeed, such a complexity trade-off relates to the effort that the robot should spend to use the knowledge for making itself able to perform the task, and the resources required for using the knowledge needed for interacting purposes. In this thesis, tools and development guidelines to design software architectures are proposed for managing memories that have been structured through interaction. Our approach can be used in different scenarios, and we illustrate it through a referencing tabletop application, where we took into account also such a trade-off.

1.1 Background

To perform non-physical HRI, the robot should be able to cope with qualitative knowledge, since we do not want to force users to rely on quantitative information only. A simple, but challenging, scenario concerns spatial relation between objects in a table, *e.g.* on the right-hand side of, or close to. If we consider two agents that communicate for aligning their beliefs before performing a joint task, many uncertainties arise already in a simple spatial domain. For instance, a common reference frame is required, as well as a definition of the orientation of objects, whose shape might also influence the semantics of spatial relations.

Although our architecture is not explicitly meant for maintaining memories of spatially related objects only, we deployed it in this scenario since it is simpler to validate than other domains, such as the one involving affordance or temporal relations for instance. In particular, we consider a robot and a user at the opposite sides of a table where objects are spatially related and identified through some qualitative properties, such as colour and shape. Specifically for this scenario, we collected the responses of 190 volunteers that filled a survey[1.1], where pictures showing objects on a table were proposed. Without providing an explicit reference frame and a definition of spatial relations, we asked to reply positively or negatively to simple statements about the pictures, *e.g.* "a ball is behind another". We

1.1 https://docs.google.com/forms/d/e/1FAIpQLScoP-qfk8Nx2MmTKMNJx6YdRaswJGvR39jTiXnj91wu79Wwew/viewform.

observed that in 17 questions over 31, the average score was of 50 ± 15%, which highlights a profound disagreement among the volunteers.

If the robot has to operate in this scenario, it would necessarily be able to adapt to each person, but it might be the case that the representation detail of the pictures we proposed was too abstract and led to ambiguities. Therefore, we propose to other 140 volunteers a similar survey[1.2], but with the possibility to give a rate of the agreement through a 7-item scale. In this case, we did not notice any statistical differences between some clusters of volunteers based on their native language. Moreover, we observed that the distribution of agreement among persons was not Gaussian but it could be used for characterising the scene with a considerably lower degree of ambiguity. This implies that with vague spatial relations, the robot could reason on a fuzzy semantic that is familiar to many different users. However, finding such a common representation for encoding knowledge in the robot memory is far being trivial due to the complex behaviour of the users, even if we considered one of the simplest scenarios.

1.2 https://docs.google.com/forms/d/e/1FAIpQLScF6fzYWOKvLH3BJS1G7kP1lp9K6kcGQ1VRxwWx_J3jMLhiPw/viewform.

In a typical development of Human-Computer Interaction (HCI) applications, where the problem is to find the most suitable ways for representing symbols in a desktop through which person interacts, the iterative development process [82] is often used. Such a development process is specifically designed to take into account unpredicted behaviours of the users, which are involved already from the early stage of the development for providing feedbacks. This process is made of three phases that are iteratively performed until the end of the development. The fist consists of developing a part of the system, in the second experiments are performed, and in the third feedbacks are evaluated form planning the first phase of the next iteration. Remarkably, during the latter phase, small system's changes should be scheduled in order to make the user's feedbacks guiding the development as much as possible, *i.e.* the more iterations are done during the development process, the better the interaction is likely to be.

We aim to use the iterative development process for guiding the design of the representation that a robot should have for performing specific tasks jointly with a person. In this way, we do not only want to investigate the trade-off among different levels of representation details to perform a specific task efficiently, but we also aim to use the users' feedback for building a robot that can share its beliefs with the purposes of improving the interaction. Remarkably, this would also be a possible system for addressing the opposite problem, *i.e.* the cases where the user wants to explain its beliefs to the robot for improving the interaction. For doing so, we require to perform as many changes to the system and experiment as possible, with the purpose of taking comparable measurements. Such a complex development process should be supported from a modular system, that allows for extensions and rearrangement of its components. Moreover, since we

are supposing to involve users already from the initial stage of the development, the system should be completed as far as experiments for a particular iteration are concerned. Unfortunately, if we consider a memory that evolves through interaction, performing experiments with the Wizard of Oz paradigm is not trivial since we require to annotate a large amount of knowledge.

We considered ourselves at the beginning of the first development iteration, and we want to develop an architecture that can be easily extended for supporting the following iterations. Therefore, we do not focus only on evaluating the interactions, but also assessing the features of the architecture we used. In other words, we propose an architecture that can be used as a guideline for developing knowledge representation for other HRI applications. We based our architecture on the Unified Modeling Language (UML), and we implemented it on the Robotic Operative System (ROS). As we mentioned, the architecture represents knowledge in OWL ontologies since it is a general-purpose standard, suitable for representing and reasoning on the HRI domain introduced above. However, we experienced a lack of integration between OWL and ROS that we had to tackle for implementing a flexible architecture for supporting iterative developments.

Indeed, some drawbacks make developers reluctant to include ontology in their architectures [17]. In particular, OWL implies a shift of developing paradigm, which is not compatible with Objected Oriented Paradigms (OOP). Thus, the developer not only should design parts of the architecture using a different paradigm, but it should also care about their interface with the other components. The interface should be such not to limit either OWL representation and ROS performances, while at the same time being flexible to changes. Noteworthy, the memory is supposed to be shared among the entire architecture, and changes to its representation should not affect the other modules. However, this is not typically the case, since scripting paradigms are used for interfacing with an ontology. Also, another drawback is due to the reasoning complexity, which typically scales exponentially with the size of the ontology. For this reason, only very small ontology can be used for representing knowledge that changes frequently, but in most of the application, a simpler and faster lower layer is recommended to perform computations at high frequency.

1.2 Objective

In the thesis, we discuss the advantages and drawbacks of an architecture that relies on symbolic representations of robotic memories. We motivate why DL is a promising formalism for the development of architectures for HRI, and we describe the issues to be taken into account while deploying OWL ontologies in a ROS architecture.

Due to such issues, state of the art architectures exploits a limited part of the potential OWL features that can be used for structuring runtime knowledge in the service of robots behaviours. Indeed, typical architectures use ontology passively, where a single component is in charge of maintaining the knowledge while exposing few functionalities to the other modules. In those architectures, the ontologies are usually devoted to reason on a static domain definition, which is carefully designed by knowledge engineers.

Nowadays, we are still investigating robots deployed in simple scenarios compared with the unstructured environments in which they are supposed to be used. In many of those situations, the knowledge the robot requires is limited, mainly due to the limited set of perception and action functionalities it can rely on. In those situations, primitive data structures are efficient and do not introduce any extra levels of complexity in the design of the architecture. Nevertheless, such a structure would sharply limit the robot if its set of functionalities increases.

Practically, consider the case in which we want to extend an architecture with a new module that produces data, which could improve the performances of other modules if the knowledge they generate is aggregated and contextualised. If the architecture relies on primitive data structures, we would need to redesign the system for updating the structures used by each component and introducing new reasoning components for aggregating purposes. We argue that if a more structured knowledge is used instead, we could design software interfaces that are extendible in terms of the semantic of the data they involve.

If we assume to have such an extendible architecture, we could think to use it for implementing different knowledge aggregation approaches, and this would be a crucial step toward a comparison and evaluation of the best reasoning techniques for a specific domain. Indeed, the state of the art systems that employ symbolic representations are mainly based on heuristic models. If they model simple and deterministic contexts, it is possible to validate their semantic behaviour based on common sense examples and counterexamples. However, if the models are supposed to represent complex contexts, involving uncertainties and structuring evolution of knowledge over time, the validation process becomes not trivial. For this reason, in complex scenarios, we argue for an iterative development process, which involves a systematic comparison of different heuristics.

The lack of comparison in the literature is mainly due to the challenging problem of defining a common framework for developing ontology-based architectures for robotics, rather than passively load an ontology inside one component. Such a framework should allow defining interfaces not only among components but also between pieces of knowledge they require and provide in/from memory. In this way,

it could be possible to integrate different reasoning techniques independently from the other components, *i.e.* we assume knowledge to be given in accordance with some interface. Consequently, it could be possible to provide each reasoner with a semantic representation of the same context and use the architecture for evaluating and tuning them. Remarkably, after this step, it would be possible to deploy different reasoners in the same architecture for being contextualised and used at runtime, as shown in Chapter 10.

With the objective of validating the feasibility of our framework, we investigated design patterns that support the development of extensible software architectures based on symbolic representation for HRI. Since we want the architecture to support the iterative development process, we built a simple but complete system first (as far as measurement and comparison are concerned), with the purpose of extending it later toward real scenarios. Therefore, we focused on a simplified tabletop environment concerning spatially arranged objects, with the objective to design architectures as flexible as possible. Then, based on an evaluation of the obtained architecture, we can conclude its possible deployment in more complex scenarios. In particular, the feature of the architecture we considered are

- *generality*, which indicates the limits of the knowledge domain it can represent and use;
- *modularity*, showing the boundaries of re-using and interfacing components in a particular domain;
- *scalability*, which concerns the performances of the system when the number of modules, as well as amount of knowledge, scale up.

To support an iterative development process for HRI, and evaluate the features of our design pattern, we consider an architecture with the following requirements.

- *Feasibility*: the computation demand should be suitable for soft real time constraints as far as HRI is concerned.
- *Shareability*: it should use mainstream tools, to allow developers to integrate as many modules as possible for perceiving, planning and acting.
- *Expressibility*: it should generically support the design of an application domain, and allow developers and users to rely on expressive semantics for representing it.
- *Explicability*: it should represent the knowledge in a way that a developer can understand its content during an iterative development process. Also, the architecture should be such to make the robot able to explain its beliefs and intentions for reducing the interaction gap.

- *Robustness*: the architecture should be able to deal with noise and uncertain knowledge.
- *Long-term*: it should be such to access and use the knowledge with a computation complexity that does not degenerate over time.

1.3 Contribution

We wanted to design an architecture in which a rich data structure is used as a central knowledge base, that distributed components use in the service of robotic behaviours for HRI applications. In particular, we consider a scenario where a user demonstrates to the robot some situations that are supposed to be interesting for an application and, though a multimodal but not physical interaction, supervises the beliefs of the robot. In particular, during a knowledge revision phase, the robot is an observer, that learns from demonstrations and allow the user to – eventually – perform knowledge refining through natural language. After such a bootstrapping phase, a structured memory of experiences is stored, and the robot is supposed to exploit the knowledge it contains for acting into the environment.

This thesis focusses on the knowledge bootstrap phase, and in the maintenance of such a structured memory for long-term. We spent effort for generalising the semantics of the knowledge that can be structured in the robot's memory, and we designed software interfaces expressive enough for being flexibly used in a ROS architecture. Although the design of perceiving and acting modules that are supposed to be based on the robot memory is not in the main scope of this thesis, for testing purposes, we had to consider a complete architecture that spans from perception to action.

In particular, we used points cloud for perceiving objects on the table as well as user's gestures, such as pointing. Also, we process natural language to engage dialogues between the user and the robot, which the only action it performs during knowledge bootstrapping is speaking. We design the bootstrapping phase such to generate knowledge that can be directly used with symbolic contextualising techniques for activity recognition or decision making, as well as for symbolic, or subsymbolic, planning.

However, describe continuous spaces is a challenging open issue, *e.g.* to be used for motion control or physical interaction, since we consider a purely symbolic representation for facilitating verbal interaction. Using Monte Carlo simulations performed over symbolic knowledge, we related qualitative symbols with a spatial semantic to 2D positions that the robot should reach. Nevertheless, apply this approach for knowledge that does not represent a spatial domain is far from trivial. In those cases, we argue for architectures that interface a high symbolic

layer to a lower layer, which evaluates functions in a numerical domain driven by the former.

In this thesis, we wanted to investigate the issues of maintaining robots' memories deeply. Therefore, we present the implementation of simple proof of concept components for visual recognition, multimodal integration, and dialogue managing components. In those components, we used simple approaches for generating qualitative data from sensors in a symbolic form, and we used it as a ground for testing the management of the memory through interactions. Although there is a large room of improvement for those components with other states of the art approaches, we used them to show how we generate the knowledge that we want to maintain in memory through sensors.

The management of the knowledge in the memory is done with the Scene Identification and Tagging (SIT) algorithm, which is the core of our knowledge bootstrapping approach. This algorithm performs one-shot learning, and structured classifications of scene demonstrated over time. The algorithm structures experiences in a logic graph and, based on fuzzy ontology, it can take into account the uncertainties of the perception modules as well as the vagueness of qualitative knowledge. We used the algorithm for making a robot able to explaining and correcting its belief based on dialogues with a supervisor. Also, we propose an architecture to store, retrieve, consolidate, and forget long-term structured experiences in contextualised parts of the memory, *i.e*. based on a semantic and episodic representation of a situation.

We tested the algorithm in a simple scenario and, through examples, we discuss in details its interface for highlighting how it can be used in different applications. In particular, we propose the concept of *semantic interface*, which represents the knowledge that a ROS component requires and provides in a knowledge base. This is different from a typical interface between components, where the type and semantic of provided and perceived data is hard coded during the architecture development. Instead in a semantic interface, a component needs to specify the syntax in which knowledge is stored, but the definition of its semantic can be delegated entirely to other components in charge of reasoning on the memory.

The deployment and test of such a novel type of interface in a robot architecture required to develop some tools for exploiting symbolic reasoning among distributed and general-purpose components. In particular, we developed a ROS service that allows distributed component to manipulate and query multiple ontologies in a thread-safe manner. The service is based in a novel API that interfaces OWL to OOP objects, and it allows to implement algorithms for managing the memory in an extensible fashion through procedure injection.

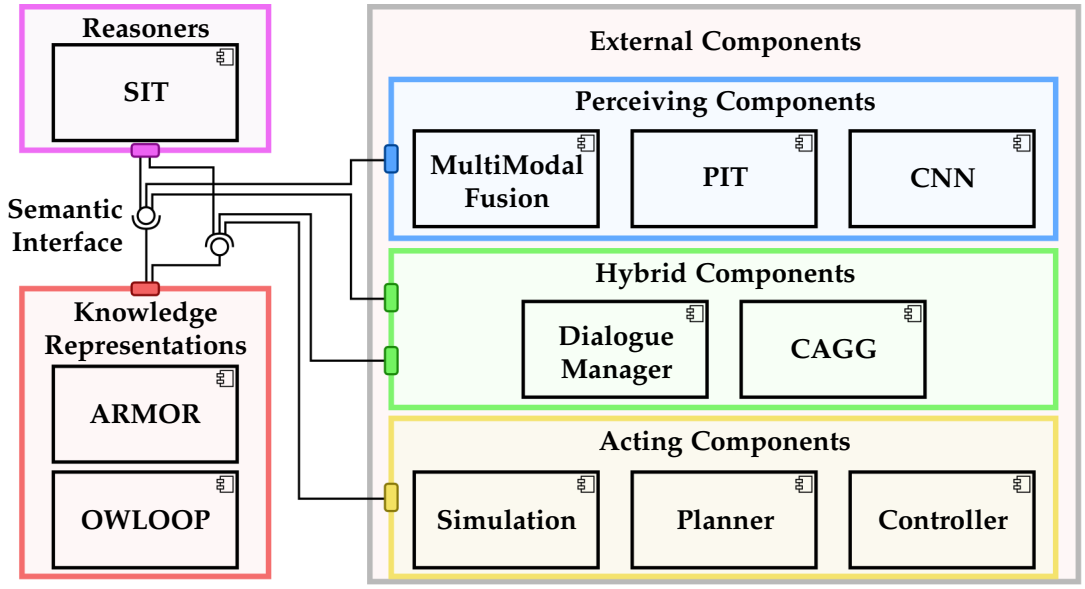


Figure 1.1: An UML-based overview of the proposed architecture for memory management.

1.4 Architecture and Thesis Structure

The thesis is organised in six parts, including this introduction (Part i) and a conclusion (Part vi). In particular, Part ii introduce some simple components that produce and use the knowledge for our testing scenario. Part iii introduces the tools we used for implementing semantic interfaces between ROS components and OWL ontologies. In Part iv we detail the algorithm that manages the memory, while in Part v we present two architectures we developed through a synergistically use of the components introduced in the previous parts. The main contribution of this thesis is presented in parts iv and vi, while parts ii and iii give a background of the overall architecture and some details about the scenario and the implementations we used in our tests.

Figure 1.1 shows an overview of the presented architecture, whose components are discussed in dedicated chapters of the thesis. It has been designed with the objective presented in Section 1.2, and discussed in Chapter 11, and its components have been tested as far as our scenario concerned[1,3]. Due to issues of interfacing OWL ontology with a ROS architecture (discussed in chapters 5 and 6), for which we had to find a solution, we could only preliminary test the overall architecture.

1.3 we provide open-source implementations at `https://github.com/EmaroLab` *with prototyping purposes.*

The architecture is composed of modules that we can be classified in the following categories.

- The *external modules* (addressed in Part ii, Chapter 9 and Appendix A) are heterogeneous concurrent components that should respect some real time constraints. They can be further categorised in

  - *perceiving modules*, which are in charge to generate knowledge,
  - *acting modules*, which reacts to particular states of the memory for driving low layer controllers, *e.g.* a planner,
  - *hybrid components*, which generate knowledge and requires feedback from memory in order to implement some acting behaviours, *e.g.* for dialogues.

- The *reasoners* (addressed in Part iv and deployed in ROS architectures in Part v) are components that elaborate the knowledge for maintaining a semantic structure of experiences over time.

- The *knowledge representations* contain semantic data structures shared among all the components of the architecture, *i.e.* the robot's memory, and provides flexible developing tools (addressed in Part iii).

We consider external components to provide (and require) knowledge from memory, which will be (or have been) elaborated from reasoners. We suppose the perception modules to proved symbolic data, *e.g.* a tuple `hasColor(x1, red)`. Acting modules should retrieve similar statements for activating some behaviour, while reasoners are in charge of evaluating models of the experiences the robot had and deduces new knowledge. In other words, reasoners aggregate data coming from perception to provide rich information for acting. For generality, we consider reasoners that retrieve inputs from memory instead of being directly interfaced to external components, which consequentially has to access the knowledge representation directly. This implies that the knowledge representation is interfaced with many, and potentially all, the components of the architecture. For this reason, the interface between components and the memory is crucial for flexible designing such an architecture. This thesis proposes to extend the definition of ROS interfaces through the specification of the knowledge that a component requires and provide in the knowledge representation.

Together with the novel type of interfaces involved in this architecture, the main contribution of the thesis is a reasoner that we tested with some simple external components. In particular, in Chapter 2, we present a stack of ROS nodes called the Primitive Identification and Tagging (PIT), which uses a Kinect to segment objects on tables and identify some symbolic properties, such as shape and colour. While in Chapter 4 we propose a multimodal symbolic fusion system that can be used to create knowledge based on a combination of PIT and neural networks for generating more robust and expressive outcomes. Also, in Chapter 3, we present the Concept Action Grammar Generator (CAGG), which evaluates sentences based on static grammars. The latter is a very simple way to perform natural language process,

but we argue that the output it provides is similar to more advanced techniques[1.4], *i.e.* semantic tagged symbols describing parts of a sentence. Based on such information we present in Chapter 9 a technique to ground such tags in the memory in order to manage dialogues between a robot and a human supervisor that might want to correct its beliefs. Last but not least, in Appendix A, we present the interfacing of external components for path planning based on Monte Carlo simulation over fuzzy symbolical representations.

1.4 e.g. https://dialogflow.com/

In the architectures proposed in chapters 9 and 10, the components introduced above interface among each other for synchronising purposes (links not shown in Figure 1.1), and with the knowledge representation for storing, retrieving or manipulate data in memory. This is done through a ROS Multi Ontology References (ARMOR), which is presented in Chapter 6 as a service for OWL manipulations and queries. It embeds the OWLOOP API, presented in Chapter 5, which is a convenient tool we designed for flexibility, integrability, and maintainability purposes. Thanks to those components we could develop architectures based on the novel concept of semantic interfaces.

In Chapter 7 we detail the SIT algorithm, which formalises the main computation we performed to generate and maintain experience in the memory. It defines a particular semantic interface that the external components presented in this thesis respect, but we describe the algorithm for a generic set of components that create knowledge with a fixed syntax, but customisable semantic. In Chapter 8, we extend the algorithm to be robust in case of uncertain environments, and we test it in a referencing scenario that we also used in chapters 7, 9 and 10.

In particular, Chapter 10 proposes an implementation of the architecture deploying several reasoners based on SIT. This architecture has been inspired by cognitive memory assessments, such as the division in semantic and episodic representations of experiences, and their association and runtime through rehearsal processes. For that architecture, we provide templates for implementing consolidating and forgetting behaviours. Remarkably, it is based on a general representation of memory *types*, which could also be used also for not cognitive architectures where contextualised knowledge representations are required.

Part II

Perceive for Acting

This part discusses some simple components we used for testing our architecture. Its contribution is marginal with respect to the other parts of the thesis, but it gives the details on how we generate symbols for encoding robot's experiences.

2

A Software Architecture for Object Perception and Semantic Representation

Abstract

In the near future, robots are expected to exhibit advanced capabilities when interacting with humans. In order to purposely understand humans and frame their requests in the right context, one of the major requirement for robot design is to develop a knowledge representation structure able to provide sensory data with a proper semantic description. This paper describes a software architecture aimed at detecting geometrical properties of a scene using an RGB-D sensor, and then categorising the objects within to associate them with a proper semantic annotation. Preliminary experiments are reported using a Baxter robot endowed with a Kinect RGB-D sensor.

Perception · Semantic knowledge · RGB-D sensor · Software architecture

Authors

Luca Buoncompagni and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

*Proceedings of the 2nd Italian Workshop on Artificial Intelligence and Robotics, a workshop of the XIV International Conference of the Italian Association for Artificial Intelligence (AI*IA 2015, Ferrara, IT).*

2.1 Introduction

Advanced human-robot interaction processes in everyday environments are expected to pose a number of challenges to robot design, specifically as far as perception, knowledge representation and action are concerned. Examples where advanced capabilities in robot cognition play a central role include robot companions [165] and robot co-workers [70], just to name a few.

It is expected that a major role in robot cognitive capabilities for human-robot interaction will be played by a tight connection between robot perception processes and their semantic representation. The latter is expected to provide robot *percepts* with explicit contextual knowledge, which is implicitly assumed to be present when two humans interact and, after an adaptation process, reach a so-called *mutual understanding* state [173].

In the long-term, we consider human-robot interaction processes where a robot and a human share a workspace, and have to interact (physically, verbally or by means of gestures) in order to perform a certain joint operation. Examples of these processes include object handovers, joint manufacturing tasks, or typical household activities. In this paper, we focus on robot perception capabilities: a robot is able to detect and track the objects present in the shared workspace and, if they belong to known categories, to provide them with semantic meaning. To this aim, the proposed software architecture provides two main functionalities:

- *Clustering, tracking and categorisation*. Starting from RGB-D data, the scene is processed to detect individual clusters in the point cloud. The position of each cluster, independently of its *shape* (in terms of the configuration of constituent points) is tracked over time. If a cluster can be mapped to a known basic geometric class (*i.e.* plane, cone, cylinder or sphere, a cube being given by six planes in a specific configuration), they are labelled accordingly using the Random Sample Consensus (RANSAC) algorithm [137].

- *Semantic description*. When object categories are determined, an ontology is dynamically updated to associate objects in the workspace (*i.e.* instances) with their semantic description (*i.e.* concepts). To this purpose, Description Logics (DLs) are used [10]. Once objects are classified, it is possible to tune robot behaviours accordingly, for example associating them with specific grasping behaviours, grounding them with proper verbal tags, using them in action planning processes.

The paper is organised as follows: Section 2.2 describes the proposed architecture; Section 2.3 discusses preliminary results; Conclusions follow.

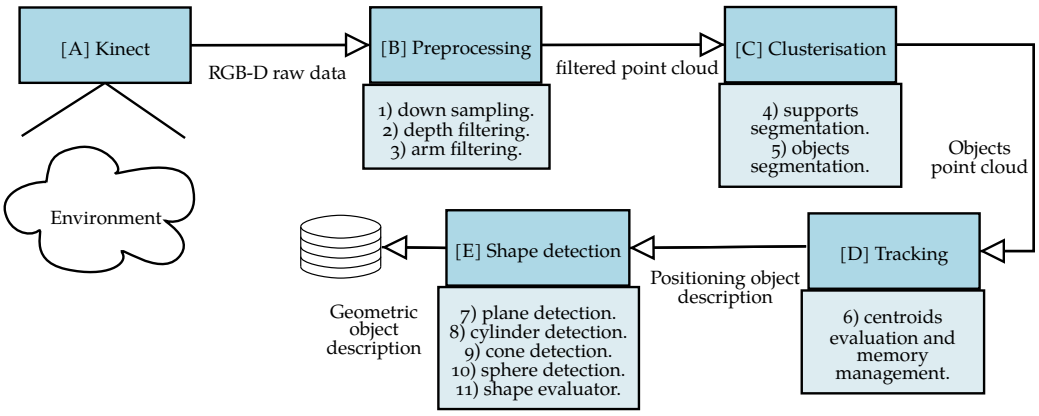


Figure 2.1: Abstract representation of the system's architecture: components (blue squares) represent information sources and computational units, exchanged data (arrows) are related to the internal information flow.

2.2 System's Architecture

The proposed software architecture[2.1] is based on the *computational* design pattern [28], *i.e.* a sequence of basic computational steps carried out in sequence according to a pipeline structure (Figure 2.1). First, a raw point cloud is acquired by the Kinect component ([A] in the Figure) and, for each scan, depth data are preprocessed ([B]). Then, the Clusterisation component ([C]) detects object supports (*e.g.* tables or shelves), and segments the objects above them, by generating a point cluster for each object. Such a *mid-level* representation is used by the Tracking component ([D]), which compares each cluster in the current point cloud with clusters already present in memory, updates their position and, if a new cluster is detected, registers it in memory. Finally, the Shape detection component ([E]) provides an estimate of the object basic primitive shape, as well as its parameters.

2.1 https://github.com/EmaroLab/primitive_identification_tracking_tagging.

The *Preprocessing* component performs a sequence of steps sequentially. First, a downsampling step is carried out to decrease the number of points in the point cloud provided by the RGB-D sensor [167]. Then, vectors normal to all the surfaces are computed [133]. The data is limited to all the points belonging to a semi-sphere around the Kinect-centred reference frame, which allows for focusing on a particular area of the workspace thereby drastically improving the overall system performance. Finally, the point cloud is also filtered to remove the points related to the robot's arms. This is done by enveloping the links related to robot arms in bounding boxes, and checking for a point-in-parallelepiped inclusion.

The *Clustering* component recursively applies RANSAC to find all the horizontal planes in the scene. This information is used to determine the points belonging to the objects located on planes (*i.e.* acting as *supports*). Finally, an Euclidean clustering algorithm is applied to segment the objects in the scene [132]. As a result, the component gen-

erates for each support a set of clusters related to objects located above it. Each cluster i is represented by its *visual centroid* $c_i^v = (x_i^v, y_i^v, z_i^v)$, computed as the mean of all the points in i.

Although many approaches to obtain a robust tracking are available (see the work in [55] and the references therein), in this preliminary work we adopted a simple geometrical approach. Our aim is to obtain and evaluate a hybrid geometric/symbolic representation of objects. Previously detected objects are stored in memory, specifically using their visual centroid and an associated list of cloud points. Our current implementation of the *Tracking* component is depicted in Algorithm 2.1. After an initialisation phase (from Line 2.1.1 to 2.1.10), first an association between current and tracked clusters is performed (from Line 2.1.12 to 2.1.15), then old clusters are removed (from Line 2.1.17 to 2.1.19). Given two clusters i and j detected at time instants t_1 and t_2, we refer to their visual centroids as $c_i^v(t_1)$ and $c_j^v(t_2)$. We assume that j

Algorithm 2.1: The tracking component.

Input : A vector C of n clusters belonging to the same support; a vector T of m tracked clusters; a matrix D of $n \times m$ distances between current and tracked clusters; a vector U of m counters to check for how many scans a cluster has not been updated; a vector F of m Boolean values to keep track of which clusters have been updated.

Const : The radius $\epsilon \in \mathbb{R}$ and the threshold $\eta \in \mathbb{N}$.

2.1.1 For each $f_k \in F$, $f_k \leftarrow false$
2.1.2 **foreach** $c_i \in C$ **do**
2.1.3 For each $i = 1, \ldots, n$ and $j = 1, \ldots, m$, $D_{i,j} \leftarrow \infty$
2.1.4 **foreach** $t_j \in T$ **do**
2.1.5 $d_{i,j} \leftarrow dist(c_i, t_j)$
2.1.6 **if** $d_{i,j} < \epsilon$ **then**
2.1.7 $D_{i,j} \leftarrow d_{i,j}$
2.1.8 **if** $\exists D_{i,j}$ *such that* $D_{i,j} = \infty$ **then**
2.1.9 create $t_k \in T$ using c_i
2.1.10 add and initialise $u_k \in U$ such that $u_k \leftarrow 0$, $f_k \in U$ such that $f_k \leftarrow 0$
2.1.11 **else**
2.1.12 $d_o = d_{i,j}^* \leftarrow argmin_{i,j}(D)$.
2.1.13 **if** $f_o = false$ **then**
2.1.14 update centroid and point cloud of t_o using weighted average between c_i and t_j
2.1.15 $f_o \leftarrow true$, $u_o \leftarrow 0$
2.1.16 **foreach** $f_k \in F$ *such that* $f_k = false$ **do**
2.1.17 $u_k \leftarrow u_k + 1$
2.1.18 **if** $u_k > \eta$ **then**
2.1.19 delete $t_k \in T$, $u_k \in U$, $f_k \in F$

%	*Empty*	*Plane*	*Sphere*	*Cone*	*Cylinder*
Empty	87.40	11.65	0.20	1.06	1.60
Plane	12.50	82.54	6.63	7.22	8.70
Sphere	0.00	0.00	93.17	0.00	0.00
Cone	0.00	0.32	0.00	66.04	0.11
Cylinder	0.10	5.49	0.00	25.68	89.59

Table 2.1: Confusion matrix describing the error distribution of the RANSAC-based shape detection.

is an updated representation of i if $c_j^v(t_2)$ is located within a sphere of radius ϵ centred on $c_i^v(t_1)$. A tracked cluster is removed from memory if it is not updated for η consecutively scans.

Finally, the *Shape detection* component associates each cluster with a possible primitive shape (*i.e.* plane, cone, cylinder or sphere, a cube being given by six planes in a specific configuration), as well as its geometrical coefficients. To this aim, we employ RANSAC to find the best fitting shape based on the relative number of points belonging to those primitives. Once a cluster i is associated with a primitive shape, its representation can be *augmented* with a shape tag (*i.e.* a category), its coefficients (*e.g.* the axis for cones or the radius for cylinders), and the *geometrical centroid* c_i^g, which is computed using the primitive shape rather than the point cloud. It is noteworthy that, in principle, c^g is more accurate than c^v, since it considers not only the visible part of the object but its full reconstruction provided by RANSAC.

Currently, knowledge about primitive shapes is maintained within an OWL-based ontology [108], where all the geometric object properties can be described. Two classes are used to model objects, namely `VisualObj` and `GeomObj`. The former models objects in the form of clusters, whereas the latter represents the associated primitive shape. `GeomObj` has a number of disjoint subclasses related to primitive shapes, including `SphereObj`, `ConeObj`, `CylinderObj`, and `PlaneObj`. Two data properties are used to describe visual and geometric centroids, namely `hasVisualCen` and `hasGeomCen`, as well as properties to describe shape-specific coefficients, *e.g.* a `SphereObj` has a radius specified using `hasGeomRadius`. As a consequence, a description corresponding to a cluster i is an instance of `VisualObj` if its property `hasVisualCen` contains a valid visual `Centroid` c_i^v, and it does not contain any valid description related to `hasGeomCen`. In formulas: $\texttt{VisualObj} \sqsubseteq \exists \texttt{hasVisualCen.Centroid} \sqcap \neg \exists \texttt{hasGeomCen.Centroid}$. A similar description holds for `GeomObj`.

2.3 Experimental Results

Two experiments have been set-up, the first aimed at evaluating the performance of the architecture in static conditions, the second on a

set-up involving a Baxter robot. The system has been implemented in ROS and the Point Cloud Library (PCL) [132].

The first experiment is aimed at estimating the errors in shape detection in a static environment with multiple supports (Figure 2.2). Acquisition is made 500 times for every shape. Results are reported in the confusion matrix shown in Table 2.1. It is possible to see that system's performance is reliable specifically for planes and spheres. Slightly lower recognition scores are present for cones and cylinders.

The second experiment focuses on the performance of the tracker as well as the visual and geometrical centroids (Figure 2.4 on the left hand side). A cone has been fixed to the Baxter's end-effector through a wire, in order to mimic a pendulum-like behaviour. When the robot moves the arm, the cone oscillates. The wire's length and the cone's mass are unknown to the robot. Figure 2.4 on the right hand side shows the tracked point cloud. It is noteworthy that the cluster does not represent the object completely, which affects the visual centroid, whereas the geometrical representation of the object allows for computing a more accurate centroid. Figure 2.3 shows the tracking of the two centroids. Intuitively, it can be noticed that the variance associated with c^v is higher than that of c^g, since the visible part of the object changes while the object oscillates. Moreover, it can be observed that between the two plots there is an offset due to the geometric properties of the real object and its visible part.

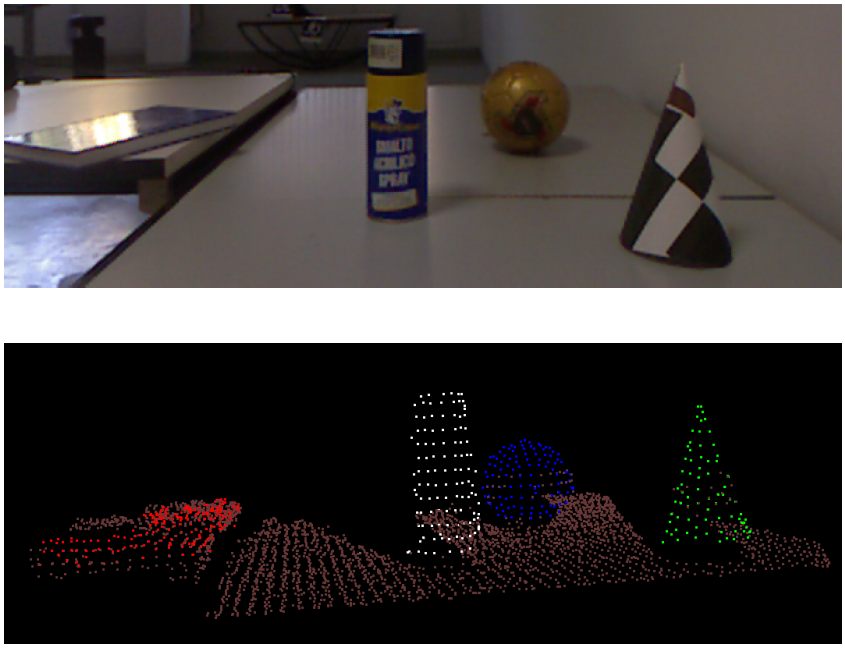


Figure 2.2: Experiment set up (left) and data visualization (right), where colours identify shapes and supports.

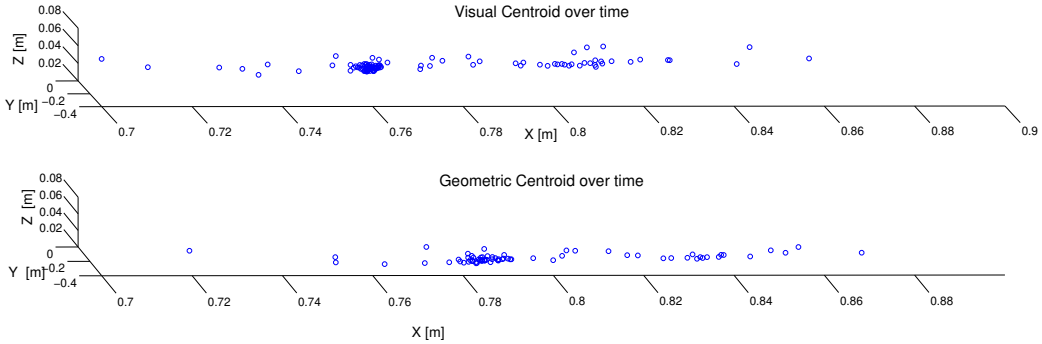


Figure 2.3: Visualisation of c^v (top) and c^g (bottom) during a tracking experiment.

2.4 Conclusions

This paper describes an architecture to model and track a few (both geometrical and semantic) properties of objects located on a table. The system is still work-in-progress. One the one hand, we are interested in exploring the possibility of using symbolic-level information to model high-level object features, such as affordances. On the other hand, we believe that the interplay between the two representation levels can be exploited to increase the overall system's capabilities.

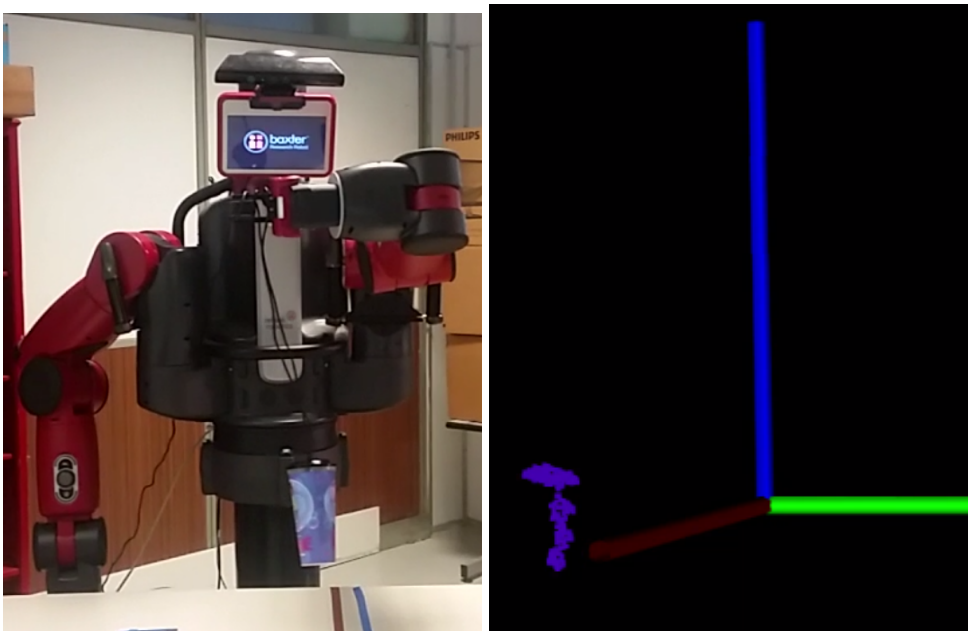


Figure 2.4: Baxter holding an oscillating cone.

3

An Open Framework to Develop and Validate Techniques for Speech Analysis

Abstract

This paper presents the Concept-Action Grammar Generator (CAGG) system, which is an *open source* framework for natural speech-based interaction to be used in the semantic analysis of sentences. The goal of our framework is two-fold: (*i*) to deliver a formalism associating a context-aware semantics to the words in a sentence; (*ii*) to provide a common framework to benchmark different implementations of evaluators with the aim of improving speech-based interaction. To this aim, CAGG includes a specifically designed formalism for grammars definition as well as parsing mechanisms to automatically create and evaluate their structure. CAGG implements a deterministic evaluator which provisional performance related to a number of natural queries is discussed in terms of correct semantic associations and computational time.

Speech analysis · Natural speech-based interface · Human-robot interaction

Authors

Luca Buoncompagni and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

*Published in the Proceedings of the 3rd Italian Workshop on Artificial Intelligence and Robotics, a workshop of the XV International Conference of the Italian Association for Artificial Intelligence (AI*IA 2016, Genoa, IT).*

3.1 Introduction

We make, as human beings, a large use of speech-based interaction in organising our daily activities, teaching to someone else how to do something, understanding other people's interests, or asking for help in difficult situations. Therefore, a *natural* and *context-aware* speech-based interaction if fundamental for human-robot interaction and collaboration as well.

There is no shortage of advanced speech-to-text systems, both as open source, *e.g.* CMU Sphinx [88] and HTK [171], and well-known commercial products such as Google Now[3.1], Microsoft Cortana[3.2], Siri[3.3] and Alexa[3.4]. On the one hand, these systems rely on formal grammar specifications, advanced data processing techniques, web searching and machine learning to provide services to their users. On the other hand, there is no widespread consensus about how to deal with the typical ambiguities we employ during communication and clarifying conversations, which are due to the common sense knowledge we imbue our speech-based interactions with. When interacting with robots, it is often the case that *contextualised* knowledge is available as *robot's belief* and it is reasonable for a robot to exploit such information to better understand a human sentence.

An open source framework to generate grammars and to easily prototype and benchmark sentence evaluators is of the utmost importance. In this paper, we propose the Concept-Action Grammar Generator (CAGG) framework, which allows for: (*i*) the context-aware semantic tagging of words in a sentence based on a specified grammar, and (*ii*) developing and validating different evaluators. Semantic tagging is the approach we adopt to combine different knowledge sources, *e.g.* the knowledge a robot may have about its environment and the context. CAGG comes with a deterministic multi-thread implementation of a grammar evaluator for a basic speech-based interaction, and it is available open source[3.5].

3.1 https://w3c.github.io/speech-api/.

3.2 https://support.microsoft.com/en-us/help/17214/windows-10-what-is.

3.3 url: http://www.apple.com/ios/siri/.

3.4 https://developer.amazon.com/alexa.

3.5 https://github.com/EmaroLab/concept_action_grammar_generator.

3.2 System's Architecture and Information Flow

The CAGG framework adopts an *ad hoc* language based on the Backus-Naur Form syntax [58]. The language defines a grammar with *rules* and *sub*-rules (identified by < and >), which represent a logic expression as a binary tree. Rules have a head and a tail, separated by :.

Listing 3.1: An example of a grammar in the CAGG syntax.

```
3.1.1 ! start <M>;
3.1.2 <M>: <R1> {@?;@∼;} | <R2> {@T;@bb;};
3.1.3 <R1>: (hi | hello);
3.1.4 <R2>: !optional(good{@!T;}) bye | !repeat(bye{@!!;},1,2);
```

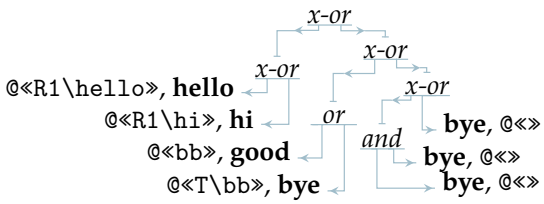


Figure 3.1: The Semantic expression Tree (ST) generated from the grammar shown in Listing 3.1.

Within rules, a number of context-aware *directives* can be introduced, denoted by `!`. In particular, `!start` specifies the root of an expression, `!optional` identifies its optional parts, and `!repeat` enables looping over an expression within a lower and an upper bound on the range. Each line always ends with `;`. Sub-rules or *terms* (*i.e.* words in a sentence, which are an expression's leaves) can be aggregated using: an empty space, `' '` (to denote an *and* logical operator), or `|` (which is a *x-or* operator). It is noteworthy that CAGG also supports C-like comments and annotations. Each rule or term can be semantically tagged using `{@...;...}`, which specifies an array of symbols *activated* in a verified parsed path. Special tags can be used using `?`, which is replaced during compilation with the rule's head or term. To indicate that the last tag must be *augmented* with the path's leaf, one can use `~`. Furthermore, a tag can be removed with `!`, while `!!` clears the tag list.

In Listing 3.1 shows a simple example of the CAGG syntax, which defines a grammar having at least a solution containing the tags: `R1` if the input is "hi" or "hello", as well as «`T\bb`» if "bye" is given or «`bb,T\bb`» for "good bye", whereas "bye bye" returns «». Accordingly, a tree representation is sketched in Figure 3.1.

Figure 3.2 shows the CAGG architecture. In order to build a grammar evaluator, CAGG implements a parsing procedure (based on ANTLR [117]) able to generate an Abstract Syntax Tree (AST) for each rule. Each AST is transformed in a binary Expression Tree (ET) via syntax manipulation and, finally, the *Grammar Generator* module builds a unique Semantic Tree (ST) (*e.g.* Figure 3.1) by associating semantic tags to leaves (*i.e.* words) and all composition of rules to tree paths. The process can be supervised through a Graphical User Interface and stored in memory.

When audio information is given, a speech-to-text interface (in the current system we use the Google Speech API [141]) is used to determine the *most likely* corresponding text string. The string must be formatted. A recursive function generates different Semantic Tree Instances (STIs). In each instance, leaf nodes are initialised with a Boolean value by matching the word with the corresponding leaf in the ST. Each STI represents a possible solution that must be feasible in terms of words consequentiality and robustness with respect to non modelled words. Our current implementation searches for all the possible solutions recursively reducing the sentence by removing the

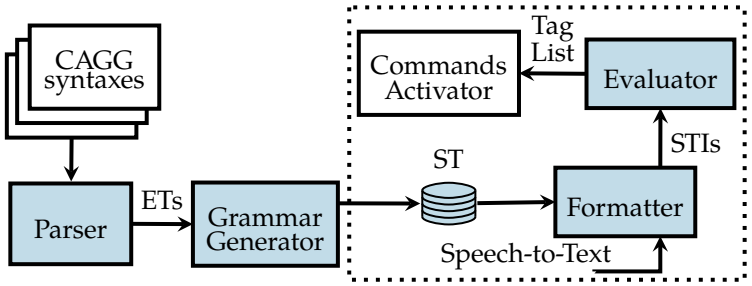


Figure 3.2: The CAGG flow to create grammars for on-line speech recognition (highlighted in a dashed box): *blue* boxes identify available modules, *white* boxes are application-specific modules.

first words until the sentence itself is empty. In this procedure, the *Formatter* module searches for the occurrences of each word in the leaves and manages possible multiple usages of the same term (*e.g.* "bye bye" in Listing 3.1 and Figure 3.1) using a depth first approach. It is noteworthy that the implementation of such an algorithm strongly affects the overall system's performance thus, current work is focused on exploring other approaches. Again on Figure 3.2, as soon as a new STI is generated it is provided to the *Evaluator* module, which computes the logical expression from the leaf to the root, by adopting an *any-time* approach. If the root is in true state, the solution is satisfied and all the semantics tags related to the true leaves are returned as an array. Finally, the Commands Activator module is in charge to map such an array of semantic symbols into application-based services.

CAGG proposes a framework for the analysis and evaluation of different approaches to grammar definition and evaluation, and provides a method to event-based action generation which employs the analysis of sentences, by means of the tags array. It allows designers and engineers to implement new *Formatter* or *Evaluator* modules based on the ST-based representation.

Listing 3.2: A grammar for simple questions with a limited set of worlds.

```
3.2.1  !start <MAIN>;
3.2.2  <MAIN>: !optional(<WH>) <QUERY> !optional(<OBJ>{@Obj~;});
3.2.3  <WH>: what | where;
3.2.4  <QUERY>: <AUX>{@?~;} <OBJ>{@Subj~;} <VERB>{@?~;} | <VERB>{@?~;}
         <OBJ>{@Subj~;};
3.2.5  <BE>: am | are | is;
3.2.6  <HAVE>: have | has;
3.2.7  <BE-HAVE>: <HAVE> | <BE>;
3.2.8  <AUX>: do | does | <HAVE>;
3.2.9  <VERB>: <BE-HAVE> | seen;
3.2.10 <OBJ>: <PRONOUN> | <THING>{@?;} | <CONCEPT>{@?;};
3.2.11 <THING>: robot | door;
3.2.12 <CONCEPT>: name | surname;
3.2.13 <PRONOUN>: I | you;
```

Do you have a name?	0.463 *sec*	«AUX\do;Subj\you;VERB\Have;Obj\CONCEPT\name»
What have you seen?	0.238 *sec*	«AUX\have;Subj\you;VERB\seen»
Are you a robot?	1.161 *sec*	«VERB\are;Subj\you;Obj\THING\robot»
Are you a machine?	0.402 *sec*	«VERB\are;Subj\you»
You are smart.	1.520 *sec*	«»

Table 3.1: Examples of system usage using the grammar in Listing 3.2.

3.3 Examples

We discuss here a simplified grammar to analyse questions posed in English (Listing 3.2). Table 3.1 contains a few examples, the corresponding computation time and the resulting sentence analysis. We see how the same word can be interpreted according to different semantics on the basis of the question and where it is in the parsing tree. Moreover, CAGG can deal with unknown words, which can be identified for further assessments. It is possible to characterise the worst case computation time, corresponding to a full parsing. This is very important for real-world human-machine interaction scenarios.

It is noteworthy that the *Evaluator* module may find also other solutions based on the grammar's definition. For example, in the grammar shown in Listing 3.1, the input "bye" could generate solutions without any semantic tags (in place of «bb»). This can happen when paths to such a term are verified. As an example it would not happen if the directive `!repeat(bye{@!!;},2,2)` were used, *e.g.* Figure 3.1. An idea to overcome this is to enable the *Command Activator* module to improve sentences analysis over time, where the improvement is related to the number of identified tags.

3.4 Conclusions

We propose the Concept-Action Grammar Generator (CAGG) system, an open source framework for natural speech-based interaction providing an efficient semantic sentences analysis. CAGG provides an open infrastructure to easily prototype and implement grammar evaluators for benchmarking purposes. Current work is focused on the integration with a system for spatial reasoning in human-robot interaction scenarios (described in Chapter 7).

4

A Software Architecture for Multimodal Semantic Perception Fusion

Abstract

Robots need advanced perceptive systems to interact with the environment and with humans. Integration of different perception modalities increases the system reliability and provides a richer environmental representation. The article proposes a general-purpose architecture to fuse semantic information, extracted by difference perceptive modules. Therefore, the article describes a mockup implementation of our general-purpose architecture to fuse geometric features, computed from point clouds, and Convolution Neural Network (CNN) classifications, based on images.

robot perception · multimodal perception · multimodal fusion · late fusion

Authors

Luca Buoncompagni*, Alessandro Carfì* and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

*Published in the Proceedings of the 5th Italian Workshop on Artificial Intelligence and Robotics, a workshop of the XVII International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018, Trento, IT).*

*These authors contributed equally to this work.

4.1 Introduction and Background

Multimodal perception gained much attention both for its bioinspired nature and for the benefits that can provide in terms of reliabilities and richness of the information. Indeed, the integration of multiple perception modalities can increase the reliability of shared information while adding to the final representation information exclusive of a particular modality. Robotic systems are an interesting scenario of application for multimodal perception since they typically have different sensors that can be integrated to enhance the robot understanding of the environment.

The multimodal perception paradigm requires a fusion process integrating information from all the modalities, an extensive overview of fusion techniques is presented in [9]. The fusion process can be performed at *feature* level, early fusion, or at *decision* level, late fusion [146]. In early fusion *feature* extracted from the raw data are combined and then analysed as a whole, on the contrary in late fusion outputs from all the perceptive modules are merged to obtain the final output. Both late [4] and early [53] fusion have been used in robotics for multimodal recognition of objects. Late fusion offers particular advantages in terms of modularity, each time a new sensor is installed the module processing its data can be easily integrated into the system. Furthermore, this approach encourages reusability and when a well-known technique to extract information from a sensor is available can be easily adapted to the particular use case.

To enhance modularity and reusability of code in robotic, we propose an architecture for multimodal perception using late fusion. Late fusion requires a common representation to be shared among all the module outputs. Because of its intuitiveness, we have designed a semantic representation in which each *item*, detected by the perception modules, is associated with a list of semantic *features*, which in the paper will be simply named *features*. The architecture uses features shared between different modalities to correlate *items*.

4.2 A Modular Software Architecture Overview

4.1 https://github.com/EmaroLab/mmodal_perception_fusion

The proposed architecture[4.1], shown in Figure 4.1, performs a late fusion of distinct perception *modules* resulting in a structure P, provided as output. The perceptive modules $\{M_i, \forall i \in [0 \dots m]\}$ have an unconstrained input interface I_i and a well defined output structure O_i. In particular, M_i generates a set of semantic *items* $X_{ij} \subseteq O_i$ described by *features* through a map $\langle v_{ij} \rangle^s$ that relates semantic *key* ($s \in S_i$) to a value (v_{ij}^s) (as shown in Table 4.1). Remarkably, we assume that in all key-values maps, the keys are unique and we define the set containing the semantic key of the whole system as $S = \bigcup_{i=1}^{m} S_i$. The features describing an item X_{ij} span in a subset of S, note that it might

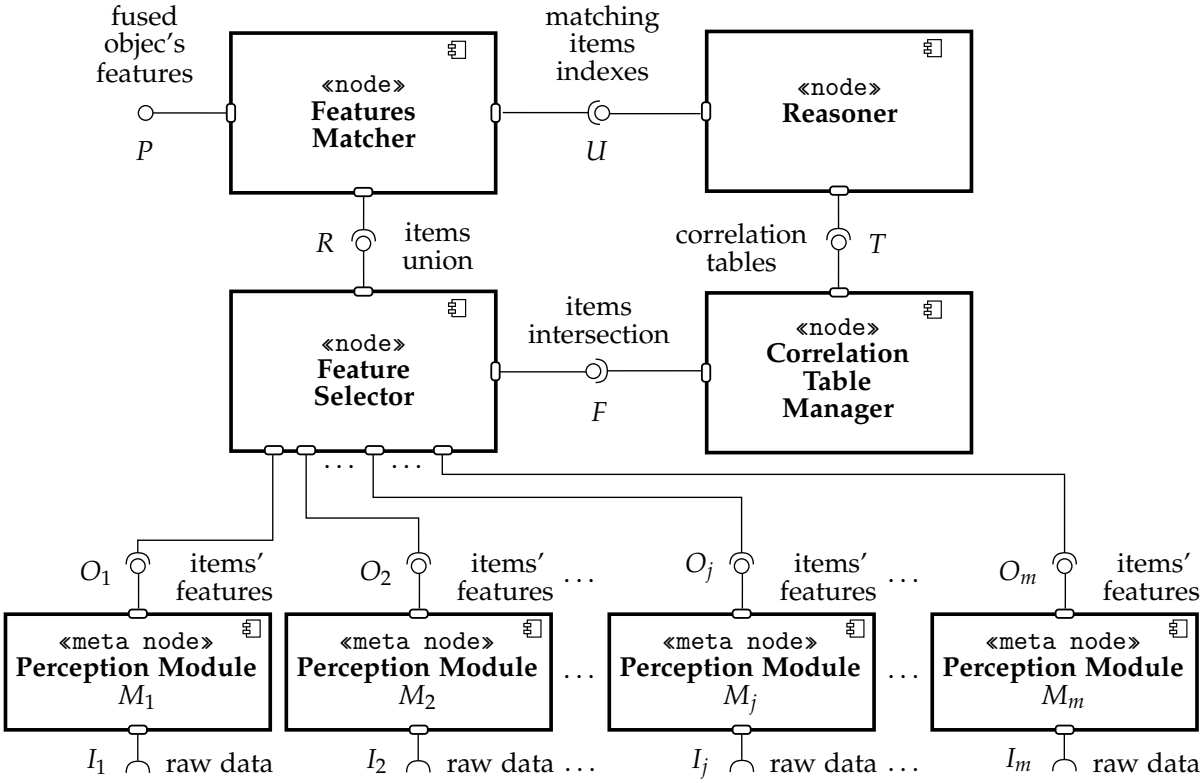
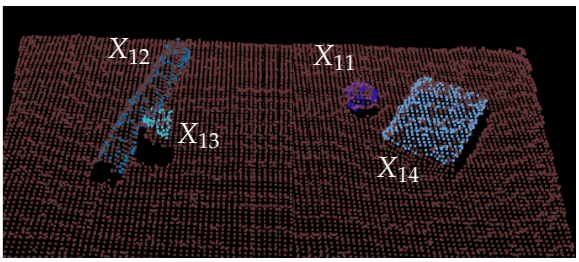


Figure 4.1: The UML diagram of the proposed architecture with m perception modules.

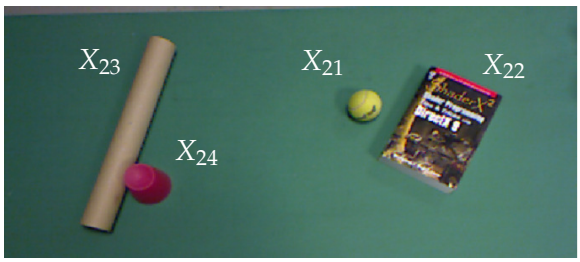
be possible $\nexists\, v^s_{ij}$. Finally, the output P has the same structure of O_i, but while the latter contains key-value maps generated from a single module, P is created by the merging process possibly using features from all the perception modalities.

The key-value structure is expressive, flexible and suitable as input for further symbolic reasoning, such as Ontology Web Language (OWL) compatible with the Robotic Operative System (ROS), *e.g.* through a bridge presented in Chapter 6. Indeed, each feature of a perceived item is represented with a semantic key, that belongs to the symbolic domain (*i.e.* is encoded as a *string*), and a value, which can be a boolean signal, a real or natural number, as well as another symbol, *e.g.* $X_{ij} = \{\langle \texttt{radius}, 0.3\rangle, \langle \texttt{cluttered}, \textit{true}\rangle, \langle \texttt{color}, \texttt{red}\rangle\}$.

The architecture interfaces with the perception modules through the Features Selector, which manages the synchronisation of the incoming data and generated R and F. Where R is the *union* of all the perceived items and F is a structure containing only the values with shared keys. The Correlation Table Manager computes the *correlation tables* T as a function of the features distance while considering only feature contained in F. This map is used by the Reasoner to identify lists of items that can be merged, and corresponding item indexes are stored in U. Finally, the Feature Matcher uses indexes store in U to fuse correlated items and provides as output a set of new items P.



(a) A point cloud, I_1, used by M_1 to extract the features of the X_{1j} items.



(b) An RGB image, I_2, (RGB image) used by M_2 to extract the features of the X_{2j} items.

Figure 4.2: An example of input and extracted features obtained from two perception modules.

4.3 Software Interfaces for Multimodal Perception Fusion

As describe in Section 4.3 the proposed architecture is designed to work with *modules* that provide outputs through the O_i interface, which is formally defined as $O_i = \{X_{ij}, \forall j \in [1 \dots \eta(i)]\}$, where $\eta(i)$ represents the number of *items* perceived by the i-th module, and each item is represented with a map of *features* $X_{ij} = \langle v_{ij} \rangle^s$. Given some output O_i from different i-th modules, we define their *union* as the concatenation of all the items perceived by all the modules, *i.e.*

$$R \doteq \bigcup_{i=1}^{m} O_i = \{X_{ij}, \forall i \in [1 \dots m], j \in [1 \dots \eta(i)]\}.$$

On the other hand, we define the *intersection* operator as the collection of pairs of items X_{hq} and X_{kp} where all the features related to not common keys are removed. And the remaining values referring to the common keys, v^z_{hq} and v^z_{kp} where

$$z \in Z_{hq,kp} = \left\{ s : \forall s \in S, \exists\, v^s_{hq}, v^s_{kp} \in R, h \neq k \right\} \subset S,$$

are structured as $H^z_{hq,kp} = \{\langle v_{hq} \rangle^z, \langle v_{kp} \rangle^z\}$. Finally the intersection is defined as

$$F \doteq \bigcap_{i=1}^{m} O_i = \Big\{ H^z_{hq,kp} : \forall z \in Z_{hq,kp}, k, h \in [1 \dots m], \\ q \in [1 \dots \eta(h)],\ p \in [1 \dots \eta(k)] \Big\}.$$

Remarkably, our architecture correlates items perceived from different modules based on feature with common semantic key. In particular, if $H^z_{hq,kp} = \emptyset$ the hq-th and kp-th items can not be directly correlated and, if $F = \emptyset$ all the items can not be correlated.

Let $\Phi = \{\varphi^z, \forall z \in Z_{hq,kp}\}$ be a set of φ^z distance functions associated to the hq-th and kp-th items; thus, each distance can be computed

as $\varphi^z\left(v^z_{hq}, v^z_{kp}\right) = d^z_{hq,kp} \in [0, \text{inf})$. We define the correlation score between the hq-th and kp-th items as

$$f_{hq,kp} = \tanh\left(-\frac{\sum_z d^z_{hq,kp}}{w}\right) + 1 \in [0,1],$$

in this way low distances values are mapped to high-level of correlation scores, and w is a parameter that can be tuned for modulate the mapping function behaviour. Through the computation of $f_{hq,kp}$ for all the pairs of perceived items in F, we obtain a set of tables $T = \left\{T_{hk}, \forall h,k \in [1 \dots m],\ h \neq k\right\}$ (thus T collects $m(m-1)/2$ tables), where T_{hk} is a table of size $\eta(h) \times \eta(k)$.

The system uses the correlation tables T as a grounded representation to reason on the best matching among the X_{ij} items. Such a reasoning generates a set $U = \{U_e, \forall e \in [1 \dots g]\}$, where g is the number of objects perceived by the architecture (*i.e.* real objects), and U_e is a list of indexes ij-th associated to the l-th items that can be merged to describe the e-th real object, *i.e.* $U_e = \langle i,j \rangle^l$. From R we extract all the l-th items $\left\{X_{ij}, \forall i,j \in U_e\right\}$ which have z-th shared and y-th unique features. Fusing the l-th items generates $P_e = \langle v_e \rangle^z \cap \langle v_e \rangle^y$, where a function δ is used to compute $v^z_e = \delta\left(v^z_{ij} \forall i,j \in U_e\right)$ and $v^y_e = \left\{v^y_{ij}, \forall i,j \in U_e\right\}$. Finally, the architecture output is $P = \{P_e, \forall e \in [1 \dots g]\}$.

4.4 Implementation

To provide an application example, we have built an implementation that uses images and point clouds to detect objects in a tabletop scenario (as shown in Figures 4.2). The architecture have been implemented using the ROS middleware, specifically for two perception modules (*i.e.* $m = 2$): M_1 and M_2. The point clouds are processed by M_1 with a stack of RANSAC simulations to segment the objects laying on the table (presented in Chapter 2). Each j-th item perceived by M_1 can be described by one or more of the features contained in $S_1 = \{\texttt{time}, \texttt{shape}, \texttt{position}, \texttt{orientation}, \texttt{radius}, \texttt{high}, \texttt{vertex}\}$. On the other hand, M_2 exploits a Convolution Neural Network (CNN) from the tensorflow tutorial [2] to detect objects and assign them a describing label. Each j-th item perceived by M_2 can be described by one or more of the features contained in $S_2 = \{\ \texttt{time}, \texttt{label}, \texttt{position}\}$. Therefore, common features of object detected by the two methods are contained in $Z_{1p,2q} = \{\texttt{time}, \texttt{position}\}$.

The correlation table T_{12} have been computed as described in Section 4.3, while the two φ^z functions have been defined as Euclidean distance. To finally merge information from M_1 and M_2 we have used an algorithm that explores T_{12} to find the row and column indexes of cells which contains a high correlation score. The algorithm ensures that each index cannot occur twice in U_r (*i.e.* each object detected

perceived items (ij)	v^s_{ij}	semantic features (s) *time* [h:m:s.ms]	*position* [m]	*shape*	*radius* [m]	*label*	...
	X_{11}	09:37:45.92	(.42, .13, .04)	`sphere`	.04		
	X_{12}	09:37:46.03	(.37, -.21, .02)	`cylinder`	.03		
	X_{13}	09:37:46.85	(.31, -.22, .03)				
	X_{14}	09:37:47.35	(.17, .34, .04)	`plane`			
	X_{21}	09:37:46.20	(.45, .11, .05)			`ball`	
	X_{22}	09:37:46.31	(.21, .33, .03)			`book`	
	X_{23}	09:37:46.37	(.34, -.19, .02)				
	X_{24}	09:37:46.42	(.31, -.22, .03)			`glass`	

Table 4.1: An example of item's features perceived through the inputs in figures 4.2a (provided in the O_1 interface) and 4.2b (provided in the O_2 interface). Perceived items are shown by row, while semantic key by columns.

from M_1 is associate at maximum to one object detected by M_2) and conflicts are addressed to prioritise higher correlation scores. Finally, to merge all the objects we have defined the δ function for `time` and `position` as the geometric mean.

4.5 Discussions and Conclusions

The paper proposed a general-purpose architecture for late semantic fusion. Indeed, it can accommodate an arbitrary set of perception modules that process different data sources, but they have to generate a specific type of outcomes, defined through the semantic item's features. Nevertheless, these semantic structures are flexible, and the architecture uses them to correlate items perceived by different modules, providing a fused representation as output.

The architecture relies on the distance between shared features, computes the correlation between items, requires a reasoner for items matching, and a function for item fusing. We deeply analysed how to orchestrate such elements in a general scenario and we present a simple implementation based on RANSAC and CNNs.

We argued that for a general case, it is required a further investigation of the distance functions between complex features, (*e.g.* `color`, `shape`, etc.), as well as regarding the types of reasoning to be performed with the computed correlation tables. On the other hand, such tables are expressive, allowing to achieve complex decisions for the item fusion. For example, they contain all the information to merge objects with partially shared features, through transitivity properties. Future developments of this work will include a wider integration of perceptive modules and an experimental evaluation of the architecture.

Part III

Represent for Reifying

This part introduces the formalism we used for implementing the memory. It describes tools for defining semantic interfaces between ROS components.

5

OWLOOP: an API to Describe Dynamic OWL Axioms in OOP Objects

Abstract

The paper presents OWLOOP, a Java-based API to interface OOP objects with OWL axioms. It supports reasoning and building of OOP structures that reflect OWL ontologies, and includes an exhaustive set of standard Description Logic (DL) semantics. OWLOOP wraps OWL API and hides the issues that are likely to occur when scripting paradigms are used to develop complex software architectures, for example, in typical robotic applications, where concurrent software are also subjected to real time requirements. OWLOOP is based on a general-purpose interface that allows synchronising a part of an ontology in an OOP domain while subjected to dynamic reasoning. The paper details such interface and discusses how OWLOOP maps OWL to OOP syntactically, thereby reflecting DL representation in OOP-like objects. We also present a simple use case to show our preliminary implementation toward the definition of a syntactic OWL-full to OOP map.

object oriented programming · ontology web language ·
application programming interface · software architecture for robotics

Authors

Luca Buoncompagni, Syed Yusha Kareem and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

Paper to be peer-reviewed.

5.1 Introduction

Description Logic (DL) [11] is one of most used formalism for representing knowledge in an ontology. Wherein, DL axioms can be asserted. Such axioms are not only retrievable through readable queries, but are also used for making implicit knowledge explicit by reasoning. From a software engineering perspective, an ontology is meant to be used in synergy with other softwares [162], that would want to read, and write, knowledge from/to an ontology.

DL is a general formalism that allows defining application domains, and general purpose upper ontologies, *e.g.* [136].The Ontology Web Language (OWL) [108] is a W3C standard implementation of the DL formalism, which has logic reasoners, an example is Pellet [144]. Such reasoners were made using the OWL API [73]. The most used reasoners perform deterministic checking, but probabilistic extensions, such as [172], as well as fuzzy representations [24], have also been proposed in the literature. Furthermore, frameworks such as learnDL [29] uses DL as an expressive interface between classifiers and other softwares that might want to use the knowledge coming from the classifiers.

Unfortunately, developers are typically reluctant to use ontologies in their repositories [17]. Due to a number of reasons, among which, one of the most relevant is that OWL formalism does not follow Object Oriented Programming (OOP) paradigm. Indeed there are many similarities between the knowledge structured in an ontology and OOP classes, methods and fields, but there are also non-trivial differences. For instance, an ontology class might have multiple parents, while in standard programming languages, such as Java or C^{++}, it is only possible to define a tree of classes. Another difference is that, in an ontology the open-world assumption is adopted, while Unified Modeling Language (UML) implicitly assumes the close-word assumption. We refer the reader to [85] for an exhaustive list of the differences between OWL and OOP.

5.2 Related Work

OWL-DL representation serves the purpose of, describing qualitative and abstract knowledge, and reasoning based on that knowledge. Such an expressive representation can be directly shared with a user. This is an interesting feature for a Human-Robot Interaction (HRI) scenario. It is being used in the domain of natural language processing, social interactions and cognitive representations, *e.g.* in [93]. Also, it allows to contextualise knowledge, and this is a crucial functionality for adaptive robots. A Large and an exhaustive ontology, such as Knowrob [18], has been used for structuring information coming from

sensors, and reasoning on the structured information (*i.e.* knowledge) for actuating specific robot behaviours.

From a software engineering perspective, several languages have attempted to reconcile OWL expressibility with the benefits of an OOP paradigm. In a survey [17] on modelling object-oriented ontologies, different OWL to OOP mapping approaches are presented. In particular, the authors categorise passive and active maps, where the latter is further divided into static and dynamic translation which, in turn, is split into dedicated, strongly-typed and dynamic languages.

On the one hand, passive usage of OWL in an OOP program, *e.g.* a Java class, is typically the approach used in robotics. In this approach, the ontology is loaded into memory, and OWL API having a factory design pattern, is used for exchanging one axiom at a time, between the ontology and a Java class. On the other hand, active OWL to OOP mapping consists of translating a serialised ontology into a static set of classes in a target language, while dynamic-active mapping translates OWL axioms into OOP classes at runtime, supporting some degree of reasoning.

Furthermore, a dynamic mapping can be based on dedicated, as well as with strong-typed, languages, but active maps based on a dynamic translation in a dynamic language seems to be the most promising approach. In particular for the latter approach, a phyton implementation of an OWL-DL to OOP mapping is presented in [89], while [87] presents a formalisation for mapping OWL-full formalism. Also, an implementation based on such an approach is presented in [12], where instances of python meta-classes have a type that reflects OWL classes, based on reasoning performed on OWL individuals and classes at run-time.

5.3 Motivation

In a robotic application, if we would like to exploit OWL-DL semantics, not only similar to a database, but also for manipulating knowledge and reasoning based on it, *e.g.* for consistency checking during the execution of a plan [32]. The fact that, we have to rely on scripting paradigms to interface with an ontology, leads to issues. Particularly, it requires us to define a high number of textual constants while accessing the ontology, which could be an issue from the software integration point of view. Also, currently, there is no possibility to interface or extend a piece of code without having to review the script, which creates an issue for code maintenance and ease in application development.

It is important to consider that a typical robotic architecture is composed of many concurrent modules, which have constraints for real time generation and consumption of knowledge, from an ontology. Since OWL reasoning is typically very complex and time-consuming

compared to the speed the requirements of robotic scenario. It is also essential to limit OWL reasoning operation (which is usually performed with a frequency). This implies that many software components have to be synchronised together within the architecture, and developing as well as debugging such an orchestra of software components based on scripting paradigms, is far from trivial.

Furthermore, it is essential to consider that a robot might have several behaviours that need to be actuated differently in different environments. This implies that while using a robot, its knowledge is often adjusted based on the components deployed in the architecture for a specific application. For this reason, having the possibility to use an OOP paradigm while designing the software components, that require an ontology's knowledge and service of its reasoner, is a crucial step towards robots in daily environments.

We do not intend to build a robot architecture from scratch since there are many modules readily available and largely evaluated (*i.e.* algorithms for planning, perceiving, and acting), that can be used in a middleware such as Robotic Operating System (ROS). In practice, dure to availability of such middlewares, a developer is not exactly free to choose the most appropriate language for his task, because it might not support complex behaviours that the robot needs; nevertheless, there are powerful tools for mainstream languages such as Java, C++ and Python. While developing a module of a robot architecture, one should consider that the module can potentially be used in different scenarios. If a module relies on an ontology, then even that module should support the same flexibility (*i.e.* ability to be used in different scenarios). Therefore the intention of our work is to provide an API that enables ease of building such modules.

5.4 Contribution

In this paper, we present the Java-based OWLOOP API[5.1], which is a wrapper of the OWL API designed to allow developers interfacing to an OWL ontology through OOP-like objects. We present an active and dynamic mapping of OWL axiom in OOP objects but, differently from other approaches in the literature, we do not attempt to define a new language that maps OWL entities to OOP objects. Instead, we design a synchronisation mechanism between copies of a fragment of the ontology in OOP objects, which are provided with a modular and common interface.

In the paper, we present a formalisation of OWLOOP considering the description of the most common OWL semantics, *e.g.* classes and sub-classes, or the definition of range and domain of some data property, as well as instances types or equivalences. With OWLOOP, it is possible to use OWL for reasoning on such semantics at runtime with

5.1 https://github.com/EmaroLab/owloop.

the objective to obtain OOP-like objects, that contain such knowledge, and that are structured among each other as in the ontology.

Although it is not clear if for very complex semantics the expressibility of our API does not degenerate, we argue that it can exhaustively map all the OWL axioms. In this paper, we show that it can largely support the developing of application in the robotic domain, also involving complex reasoning procedures. For instance, we are currently using OWLOOP for implementing a one-shot learning algorithm to represent the experiences that a robot had through the interaction of the user, which is presented in chapters 7 and 8. Also, we used OWLOOP for implementing a distributed reasoning system based on a network of ontology for recognising human activities in a smart environment [83]. OWLOOP have also been injected in the ROS Multi Ontology References service (ARMOR), described in in Chapter 6., which interfaces OWL API among the clients of robotic architectures in a safe trade manner.

Remarkably, in this paper, we do not aim to define a semantic OWL to OOP map, which remains a challenging open issue in state of the art. Instead, the paper presents a syntactic OWL to OOP map, which would reflect a DL representation in OOP-like objects, but their semantic is always accessed in a not OOP-like manner. We design OWLOOP for practical reasons, and we limited the differences that a developer would experience if the library would rely on a full OWL to OOP map, instead of a wrapper. Nevertheless, OWLOOP introduces a new layer of complexity, which might impact the performances, and this limitation must be carefully taken into account, especially for a robotic application.

As far as the presentation of OWLOOP is concerned, in the next section, we recall the DL formalism, including specific OWL definitions and behaviour of deterministic reasoners. In Section 5.6 we give an overview of the API, and in Section 5.7 we detail its main component, and how developers can use its interface. Then, in Section 5.8 we present a toy example for showing a typical design pattern for using OWLOOP.

5.5 Description Logic Primer

Let us introduce the notation and syntactic sugar that we will use in this paper. An OWL-DL ontology consists of a set of *axioms*. Each axiom is a *statement* describing a piece of knowledge, and is composed of three parts: the *expression* $\mathcal{E}$ and two *elements* x and y, *i.e.* forming a tuple $\langle \mathcal{E}, x, y \rangle$. The elements x and y could be of the following *entity types*:

- as a *class* type: (synonymous to *concept*) is a set of instances (synonymous to *individuals*). We identify it with an upper case Greek letter (*e.g.* Δ, Λ).

- as an *individual* type: which is an instance of a concept (*i.e.* class) and it has properties. We identify it with a lower case Greek letter (*e.g.* α, β),
- as a *property* type: which is a relationship between two individuals. We write them as bold upper case Roman letters (*e.g.* **P**, **R**),
- as a *literal* type: which represents data value such as strings or integers. We write them as lower case Roman letters (*e.g.* x, y)

Note we consistently maintain this nomenclature across all the chapters of the thesis. For clarity, we also specify the name of each element in an ontology with upper case letters for classes, e.g. `CLASS`, *while an individual has only the first letter capitalised,* e.g. `Individual`, *and a property has all lower case letters,* e.g. `property`.

Properties are divided into *object* and *data* properties. The former represents a relation between two individuals, while the latter between an individual and a literal, *i.e.* a number or a string. Also, the latter might be described through axioms that define their scope concerning property and its domain, *i.e.* $\mathbf{P}.\Delta$ describes that an individual α can be involved in a relation **P** with β if the latter is an instance of Δ.

DL defines several expressions $\mathcal{E}$ that are the core of the semantics of OWL-DL axioms. Some of the $\mathcal{E}$ in DL are the following: (*i*) *negation* (*ii*) *conjunction* and (*iii*) *disjunction* between classes, *e.g.* $\neg\Delta$, $\Delta \sqcap \Lambda$, $\Delta \sqcup \Lambda$; whose result is another class. As well as the (*iv*) universal, (*v*) existential, and (*vi*) bounded *cardinality restrictions*, *e.g.* $\forall\mathbf{P}.\Delta$, $\exists\mathbf{P}.\Delta$, $=n\mathbf{P}.\Delta$, $<n\mathbf{P}.\Delta$, $>n\mathbf{P}.\Delta$ (as object property restrictions), and $\forall\mathbf{P}.\mathrm{x}$, $\exists\mathbf{P}.\mathrm{x}$, $=n\mathbf{P}.\mathrm{x}$, $<n\mathbf{P}.\mathrm{x}$, $>n\mathbf{P}.\mathrm{x}$ (as data property restrictions), with $n \in \mathbb{N}$. Also, it is possible to rely on (*vii*) *inclusion* (or subsumption, which is related to implication), (*viii*) *equivalence*, and (*ix*) *definition* of classes or properties, *e.g.* $\Delta \sqsubset \Lambda$, $\Delta \equiv \Lambda$, $\Delta \doteq \Lambda$ or $\mathbf{P} \sqsubset \mathbf{R}$, $\mathbf{P} \equiv \mathbf{R}$, $\mathbf{P} \doteq \mathbf{R}$. It is possible to define also disjunction and equivalence between individuals, *e.g.* $\alpha \not\equiv \beta$ and $\alpha \equiv \beta$. Last but not the least, it is possible to describe (*ix*) *assertions* between classes and properties, *e.g.* $\Delta\colon\alpha$, and $\mathbf{P}(\alpha, \beta)$. The former axiom asserts that α is an individual of the class Δ. While the second axiom asserts that the two individuals α and β are related to each other with the property **P**.

Remarkably, only assertions involve individuals, while all the other expressions describe classes and properties. Typically, the axioms that involve individuals are collected in the Assertional Box (ABox), while the others are collected in the Terminological box (TBox), and in the Role box (RBox). Those boxes contain the interwoven description of individuals, classes and properties respectively. While the ABox could be considered as a list of axioms, the other two can be considered as acyclic graphs, where each edge denotes an implication among pairs of classes (TBox) or properties (RBox). For each graph, the root is defined as $\top$, while the opposite operator $\bot$ is defined for representing leaves.

5.5.1 Axioms Representation in Ontology

OWL is an implementation of the DL formalism, and it allows to create and manage data structures through boxes containing the symbols

defined above. Those type of data structure is an ontology, which contains a formal definition of a specific application domain. Such a definition is based on the following three different syntactic categories [113]

- *Entities* (x, y) are of different types (*i.e.* classes, properties, or individuals), and are identified through a function $\mathscr{N}$ that returns an Internationalized Resource Identifiers (IRIs), *i.e.* a name. They constitute the vocabulary of an ontology, *e.g.* a class $\mathscr{N}(\Delta) = \texttt{ROOM}$ can be used to represent the set of all rooms. Similarly, the object property $\mathscr{N}(\mathbf{P}) = \texttt{isLinkedTo}$ can be used to represent the room-corridor relationship. Finally, the individual $\mathscr{N}(\alpha) = \texttt{Room1}$ can be used to represent a particular room.

- *Expressions* $(\mathcal{E})$ represent one of the operations listed in the previous section. For example, a class expression describes a set of individuals in terms of the restrictions on the individuals' characteristics. While other class expressions might describe that a class subsume another, or that an individual is equivalent to another. Therefore, each expression can be applied only between specific entity types.

- *Axioms* are statements that are asserted to be true in the domain being described, *e.g.* $\mathbf{P}(\alpha, \beta)$, or $\Delta \sqsubset \Lambda$. In this paper, we consider a generic axiom as a tuple $A = \langle \mathcal{E}, x, y \rangle$ and, with expressibility purposes, we consider more axioms involving the same expression $\mathcal{E}$ and the same entity x in a single tuple through the definition of the *entity set* $Y = \{y_i,\, i \in [1, n]\}$. Thus, we denote a collection of axioms $\langle \mathcal{E}, x, Y \rangle$, which indeed represents n OWL axioms.

5.5.2 OWL Reasoning

OWL does not only provide a standard and expressive knowledge representation, but it has also been exploited for implementing reasoning procedures. In particular, the reasoner is an algorithm that performs consistency and instance checking by interpreting the logic description. It can react and explain inconsistencies, as well as solve queries through specific formalism as SPARQL [143], and SWRL [74].

OWL API maintains two data structures, one is *asserted*, and the other is *inferred*. The former contains axioms explicitly stated in the ontology, while the latter includes deductions, *i.e.* new axioms, that the reasoner could realise from the former. The reasoning task is a synchronisation among those two structures since it is not possible to modify the structure dedicated to the reasoner directly.

The reasoner is supposed to checks the consistency of the axioms, the properties that an individual might have, as well as the classes having an individual as an instance. Remarkably, with this procedure, the

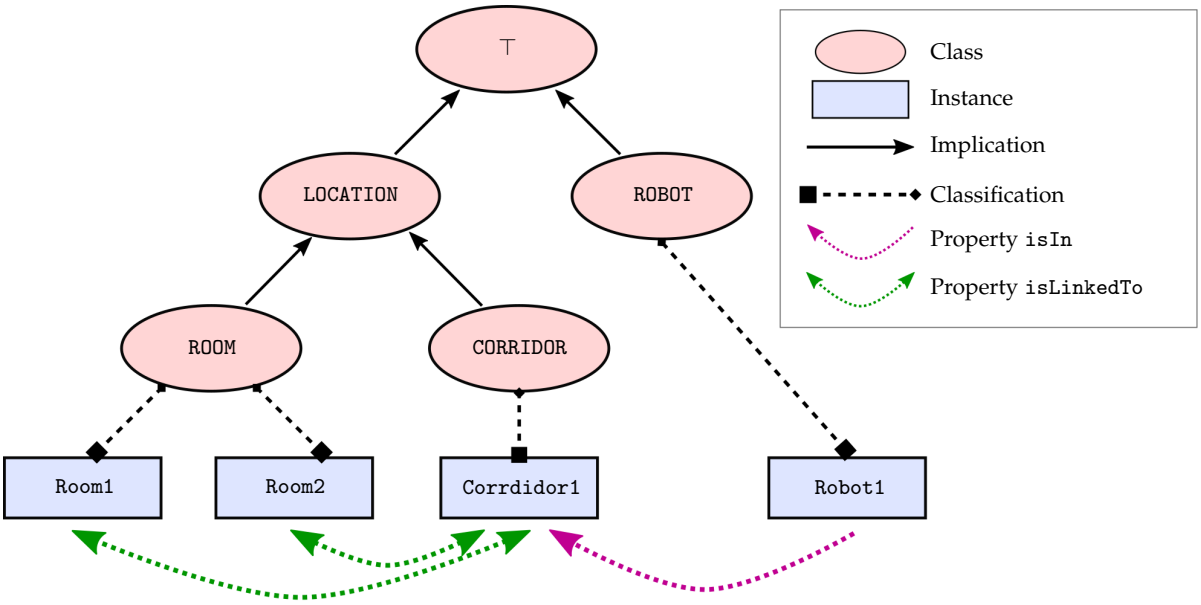


Figure 5.1: The summary of a simple ontology used for the examples in discussed in this paper.

reasoner also infers the subsumption among classes, and properties, in the TBox, and RBox, *i.e.* it completes the edges of the implications graph. Nevertheless, it cannot create new nodes due to monotonic reasoning limitations. In other words, the reasoner can create axioms that describe entities, but not entities themselves.

5.6 OWLOOP Overview

In the next sections, we support the presentation of our API with a toy example concerning a topological map of an apartment. It is based on a simple ontology which is summarised in Figure 5.1. In particular, in the TBox, there are two disjoint user-defined classes, *i.e.* `ROBOT` and `LOCATION`. The latter is in turn described through two disjoint classes, *i.e.* `CORRIDOR` and `ROOM`, which subsumes `LOCATION`. While in the ABox there are five individuals `Loaction1`, `Corridor1`, `Room1`, `Room2`, and `Robot1`, which, for simplicity, have been named as the relative classes they are instance of. Finally, in the RBox an object property is considered for representing that two locations are connected, *i.e.* `isLinkedTo`, while `isIn` describes that the robot is in a location.

Figure 5.2 shows the internal architecture of OWLOOP, which consists of two synchronised data structures. On the one hand, we consider on OWL ontology $\mathcal{O}$, which is associated with a reasoner interfaced trough OWL API. On the other, there are *descriptors* that are OOP-like objects. Each descriptor has a specific semantic that is reflected in the ontology and *vice versa* through *reading* and *writing* operations, which are based on OWL manipulations and queries.

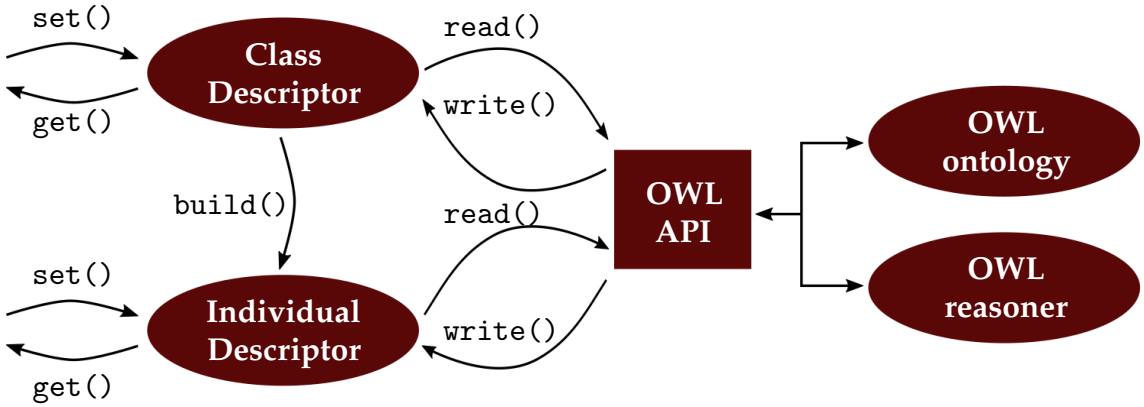


Figure 5.2: A simplified example to present an overview of OWLOOP interfaces.

Through this operation, a fragment of the ontology can be represented with an OOP fashion, and data encoded with OOP objects can be mapped in a part of the ontology as well. Therefore, the descriptor has an *internal* state that OWLOOP is in charge to retrieve or store in the ontology given some axioms. A user of OWLOOP should interact with some descriptors trough get and set like methods, and not with the ontology directly. In this way, the user can use OWL within an OOP development and, at the same time, he or she can relay on ontologies and reasoners that exploit the OWL language, but do not follow OOP paradigms.

For design OOP descriptors without limiting the expressibility of OWL ontology, we introduce the concept of descriptor *building*. During this operation, a descriptor relies on its internal state for generating another descriptor with a different semantic. For instance, consider a descriptor that representing only the classes of an individual $\mathcal{D}_1$, while another that represents the sub-classes of a given class $\mathcal{D}_2$. The first descriptor should be instantiated given the name of a specific individual, *e.g.* $\mathcal{N}(\alpha) = \texttt{Room1}$, and after a reading procedure, its internal state $\mathcal{D}_1\texttt{.get()}$ would contain $\{\top, \texttt{LOCATION}, \texttt{ROOM}\}$, if $\mathcal{O}$ is as the one shown in Figure 5.1. Instead, the second descriptor should be instantiated given the name of a specific class, *e.g.* $\mathcal{N}(\Delta) = \texttt{ROOM}$, and it would have an internal state $\mathcal{D}_2\texttt{.get()}$ equivalent to $\bot$.

As an example, let assume we want to define $\mathcal{D}_2$ based on the internal state of $\mathcal{D}_1$, *i.e.* we should find a name $\mathcal{N}(\Delta)$ that identifies the deepest class that has `Room1` as one of its individuals. With OWL API this would require to use the *factory* many times for converting names in sets of axioms, and we discussed in Section 5.3 how this leads to issues for complex architectures. With OWLOOP, it is possible to instantiate the first descriptor and use the building function for accessing the internal state of related, but semantically different, descriptors. As we will discuss more in Section 5.8, with these operations we could solve the example above by specifying only one name, *i.e.* `Room1`.

Based on the example above, $\mathcal{D}_1$`.build()` will return three descriptors $\{\mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5\}$ having as internal state some axioms about the $\top$, location, and room classes respectively. If those axioms are such to represent sub-classes (and this is defined by $\mathcal{D}_1$, as we will see in Section 5.8), it would be possible to build the obtained descriptors again to represent OOP objects about subsumed classes. For instance, the $\mathcal{D}_3$`.build()` would return all the classes in the ontology since it describes the sub-classes of $\top$. While the $\mathcal{D}_4$`.build()` returns another descriptor representing the class of rooms, and $\mathcal{D}_5$`.build()` returns a descriptor fot $\bot$, *i.e.* an empty set. Therefore, the descriptor $\mathcal{D}_5$ solves this example, and it could be such to describe other axioms for building further descriptors as well. Remarkably, each building operation does not merely return some knowledge in the ontology, but it returns a synchronisable representation of a part of the ontology, which can be interfaced and extended in an OOP fashion.

5.7 OWLOOP Descriptors

A descriptor is an object in OOP terms that (*i*) manages the synchronisation of some axioms between the ontology and its internal state, (*ii*) provides external access to such an internal state for manipulating and querying knowledge, and (*iii*) uses its internal state to build further descriptors. Due to the latter point, and since a general OWL axiom involves heterogeneous entities, we need to design a general interface for building descriptors that represent the expression axioms presented in Section 5.5.

For this reason, we designed the *abstract* descriptor as a modular interface that allows to specifying an expressive object by injecting procedures for describing specific sets of axioms, *e.g.* sub-class expression axioms or disjoint expression axioms. During the design of this interface, we consider performances issue, and we design OWLOOP to define modular descriptors that should be used to synchronise only the axioms required for an application. We will discuss this design in Section 5.8, but first we present the internal state and the interfaces of an abstract descriptor in sections 5.7.1 and 5.7.2, while in Section 5.7.3 we discuss some *concrete* descriptors.

5.7.1 Descriptors Grounding

Each descriptor is featured with an ontology $\mathcal{O}$, its internal state contains a set of axioms $\langle \mathcal{E}, x, Y \rangle$. $\mathcal{E}$ is the expression defining the axioms, X is the *ground*, which is an entity $\mathcal{E}$-related to each element of the *entity set* Y, that are all of the same type, *e.g.* either n classes or n individuals. Remarkably, the type of x and y_i must be consistent for each different expressions and, after the initialisation of a descriptor, we assume $\mathcal{E}$, x, and $\mathcal{O}$ to be immutable for such a descriptor.

For instance, if we consider an axiom based on the inclusion expression, *e.g.* $\Delta \sqsubset \Lambda$, all the elements of the entity set, as well as the ground, must be classes, *e.g.* `ROOM`. While, if we consider a class assertion axiom, *e.g.* $\Delta\colon a$, then we can define a descriptor where X is a class, and Y contains individuals, *i.e.* given a class we describe all its instances. Alternatively, it is possible to define a descriptor where X is an individual, and Y contains classes, *i.e.* given an individual we describe all its types.

As we will discuss in the following sections, such an interface is general enough to span all the axioms in an ontology, and we exploit such a generality for representing descriptor of descriptors. In particular, we extended the definition of the descriptor above in order to have an internal state containing a set of set of axioms. Formally, we collect the internal axioms as $S = \{A_k, k \in [1, m]\}$, where each A_k is a tuple containing $\mathcal{E}$, x, and Y, whose are maintained by m specific sub-descriptors. A descriptor inherits the ontology and ground of all its sub-descriptors, which must be consistent, *i.e.* all the internal axioms of sub-descriptors involve the same grounding entity x in the same ontology $\mathcal{O}$.

5.7.2 The Abstract Descriptor

While in the previous section we presented the *attribute* of a descriptor, in this section we discuss its *methods* for defining the interface of an abstract descriptor. In particular, each descriptor should access its internal state, and communicate with an ontology and its reasoner, in order to implement the following methods

`init(`$\mathcal{O}, x$`)` construct a descriptor of internal axioms involving a certain expression $\mathcal{E}$ in an ontology $\mathcal{O}$,

`void set(Y)` changes the internal state,

`Y get()` returns the internal state,

`Y query()` returns the axioms from the ontology without changing the internal state (it is also internally used to compute other methods)

`void read()` modifies the internal state to be consistent with the axioms in the ontology,

`void write()` modifies the ontology such to contain the axioms consistently with the internal state,

$\mathcal{D}$`[] build()` returns a set of n specified descriptors that represent axioms for each y_i element of the entity set.

Remarkably, OWLOOP dot automatically updates the reasoner during the computation involved on those methods. This because we want to

Ground x	**Expression** $\mathcal{E}$	**Entity Set** $y_i \in Y,\ \forall i \in [1,n]$	**Internal State** A_k	**OWL Axiom** $y_i \in Y,\ \forall i \in [1,n]$	**ID**
`OWLClass` (Δ)	*Equivalent*	`OWLClass` (Λ_i)	$\langle \equiv, \Delta, \Lambda_i \rangle$	$\Delta \equiv \Lambda_i$	$\mathcal{D}_{11}$
	Disjoint	`OWLClass` (Λ_i)	$\langle \not\equiv, \Delta, \Lambda_i \rangle$	$\Delta \not\equiv \Lambda_i$	$\mathcal{D}_{12}$
	Subsume	`OWLClass` (Λ_i)	$\langle \sqsubset, \Delta, \Lambda_i \rangle$	$\Delta \sqsubset \Lambda_i$	$\mathcal{D}_{13}$
	Imply	`OWLClass` (Λ_i)	$\langle \sqsupset, \Delta, \Lambda_i \rangle$	$\Delta \sqsupset \Lambda_i$	$\mathcal{D}_{14}$
	Classify	`OWLIndividual` (α_i)	$\langle :, \Delta, \alpha_i, \rangle$	$\Delta : \alpha_i$	$\mathcal{D}_{15}$
	Definition	`OWLOOPRestriction` $(\langle q, \mathbf{P}, \Lambda \rangle_i)$	$\langle \bullet, \Delta, \langle q, \mathbf{P}, \Lambda \rangle_i \rangle$	$\Delta \doteq \bigsqcap_{i=1}^{n} \{ q_i \ \ \mathbf{P}_i \bullet \Lambda_i \}$	$\mathcal{D}_{16}$

Table 5.1: The mapping of OWL class expression axiom in an OWLOOP descriptor of a class, which is described through axioms based on any combinations of the row of the table.

limit the reasoning times since it strongly affects the performances of a system using OWLOOP. Therefore, those methods might require to also update the reasoner in order to write or read from a fresh inferred ontology.

Let us consider a case where a descriptor $\bar{\mathcal{D}}$ is made of m sub-descriptors $\mathcal{D}_k$. In this case, we design the interface in such a way that $\bar{\mathcal{D}}$`.read()` call the read method for all $\mathcal{D}_k$, and the same also occurs for `write()`. Instead, for the other functionalities, we generate specific methods, *e.g.* $\bar{\mathcal{D}}$`.get`$\mathcal{D}_k$`()` or $\bar{\mathcal{D}}$`.build`$\mathcal{D}_k$`()`, which consider only the related part of axioms in the internal states, *i.e.* A_k.

5.7.3 Concrete Descriptors

Tables 5.1, 5.2, and 5.3 shows the definition of some descriptor for classes, individual, and properties respectively. Each table shows the parameter $\mathcal{E}$, x, and Y involved in a descriptor for representing axioms with a specific semantic. In particular, it is possible to see that in the same table different expression share the same ground, but specific $\mathcal{E}$ and Y are involved. Nevertheless, the interface introduced in the previous sections is always respected. For simple axioms, we could directly use OWL objects for defining the type of each element of the entity set, but when more complex expressions are involved, we need to design specific data structures for representing entities y_i. Nevertheless, the ground x is always an OWL object.

Let for instance consider the *classify* expression define by OWLOOP as a descriptor able to synchronise the axioms that define the classes in which an individual is an instance of, *i.e.* `OWLClassAssertionAxiom`, in Table 5.1. In this case, the ground is a class Δ, which name is given during the construction of the descriptor, while the elements of the entity set are n individuals α_i. Therefore, through the `set()` method it is possible to change the internal state by adding individuals that can be stored in the ontology through the `write()` method. Alternatively, if we set an empty entity list and we use the `read()` and `get()` methods

Ground x	**Expression** $\mathcal{E}$	**Entity Set** $y_i \in Y,\ \forall i \in [1,n]$	**Internal State** A_k	**OWL Axiom** $y_i \in Y,\ \forall i \in [1,n]$	**ID**
`OWLIndividual` (α)	*Equivalent*	`OWLIndividual` (β_i)	$\langle \equiv, \alpha, \beta_i \rangle$	$\alpha \equiv \beta_i$	$\mathcal{D}_{21}$
	Disjoint	`OWLIndividual` (β_i)	$\langle \not\equiv, \alpha, \beta_i \rangle$	$\alpha \not\equiv \beta_i$	$\mathcal{D}_{22}$
	Type	`OWLClass` (Δ_i)	$\langle :, \alpha, \Delta_i \rangle$	$\Delta_i : \alpha$	$\mathcal{D}_{23}$
	Property	`OWLOOPLink` $(\langle \mathbf{P}, \beta \rangle_i)$	$\langle (), \alpha, \langle \mathbf{P}, \beta \rangle_i \rangle$	$\mathbf{P}_i(\alpha, \beta_i)$	$\mathcal{D}_{24}$

Table 5.2: The mapping of OWL individual expression axiom in an OWLOOP descriptor of an individual, which is described through axioms based on any combinations of the row of the table.

it is possible to obtain the `OWLNamedIndividual` that are classified in the given ground Δ.

Remarkably, the reading and writing operations generate only the required changes in the ontology and the internal state. Such a change might require to update an entity in the ontology, and OWL API needs to query and delete the axioms describing the old entity before to add a new entity. Therefore, the knowledge that the reasoner infers from the ontology is not only considered during reading operations, but it might also be queried while writing.

For the *definition* and *property* OWLOOP expression axioms, in tables 5.1 and 5.2 respectively, we design a specific object to be contained in the entity set. In particular, `OWLOOPRestriction` is a tuple $y_i = \langle q, \mathcal{P}, \Delta \rangle_i$ that represent the q_i restriction of the $\mathcal{P}_\rangle$ property in the Δ_i class, where q could either be an existential ($\exists$), or an universal ($\forall$), or a bounded ($<n$, $=n$, $>n$) quantifier. An example of `OWLOOPRestriction` could be $y_1 = \langle \forall, \texttt{isLinkedTo}, \texttt{LOCATION} \rangle$ and $y_2 = \langle >1, \texttt{isLinkedTo}, \texttt{ROOM} \rangle$, where y_1 describes a property `isLinkedTo` that spans only in the instances of `LOCATION`, *i.e.* the domain- While y_2 describes that the ground must have at lest two properties `isLinkedTo` with two disjoint `ROOM`s. We consider the conjunction of each element of the entity set, *e.g.* $y_1 \sqcap y_2$, which could be the definition of the `CORRIDOR` class, *i.e.* the ground.

Similarly, we designed the `OWLOOPLink` object to represent a tuple $\langle \mathcal{P}, \beta \rangle_i$, which contains a property $\mathcal{P}$, and an individual β_i. As an example, consider a descriptor specialised to represent object property expression, in Table 5.2, with axioms having as a ground the individual `Corridor1`. In this case, an element of the entity set could be $\langle \texttt{isLinkedTo}, \texttt{Room1} \rangle$, which represents in the ontology the axiom `isLinkedTo(Corridor1,Room1)`. We adopted the same representation also for data properties with the difference that β_i is a concrete value instead of an individual, *e.g.* a number or a text. Remarkably, Table 5.3 is consistent for both data and object properties.

We mentioned that a descriptor could be made of multiple sub-descriptors, but we also assumed that the ground and the type of expression is immutable for the same descriptor. Therefore, each

Ground x	**Expression** $\mathcal{E}$	**Entity Set** $y_i \in Y,\ \forall i \in [1,n]$	**Internal State** A_k	**OWL Axiom** $y_i \in Y,\ \forall i \in [1,n]$	**ID**
`OWLProperty` (**P**)	*Equivalent*	`OWLProperty` ($\mathbf{R}_i$)	$\langle \equiv, \mathbf{P}, \mathbf{R}_i \rangle$	$\mathbf{P} \equiv \mathbf{R}_i$	$\mathcal{D}_{31}$
	Disjoint	`OWLProperty` ($\mathbf{R}_i$)	$\langle \not\equiv, \mathbf{P}, \mathbf{R}_i \rangle$	$\mathbf{P} \not\equiv \mathbf{R}_i$	$\mathcal{D}_{32}$
	Subsume	`OWLProperty` ($\mathbf{R}_i$)	$\langle \sqsubset, \mathbf{P}, \mathbf{R}_i \rangle$	$\mathbf{P} \sqsubset \mathbf{R}_i$	$\mathcal{D}_{33}$
	Imply	`OWLProperty` ($\mathbf{R}_i$)	$\langle \sqsupset, \mathbf{P}, \mathbf{R}_i \rangle$	$\mathbf{P} \sqsupset \mathbf{R}_i$	$\mathcal{D}_{34}$
	Domain	`OWLClass` (Δ_i)	$\langle \top\exists, \mathbf{P}, \Delta_i \rangle$	$\exists \mathbf{P} . \top \sqsubseteq \Delta_i$	$\mathcal{D}_{35}$
	Range	`OWLClass` (Δ_i)	$\langle \top\forall, \mathbf{P}, \Delta_i \rangle$	$\top \sqsubseteq \forall \mathbf{P} . \Delta_i$	$\mathcal{D}_{36}$

Table 5.3: The mapping of OWL property expression axiom in an OWLOOP descriptor of a property, which is described through axioms based on any combinations of the row of the table.

descriptor can have sub-descriptors all sharing the ground, since they must coincide. In other words, a descriptor can contain any combinations of sub-descriptors spanning from the same table only, with at most one for each type shown by raws. We used this modular design for increasing the expressibility of the abstract descriptor and, at the same time, limiting as much as possible the number of axioms that are synchronised by OWLOOP.

5.7.4 Descriptors Building

Let a descriptor $\bar{\mathcal{D}}$ which contains some sub-descriptors $\mathcal{D}_k$. The building method is specified for each sub-descriptor, and $\bar{\mathcal{D}}$`.build`$\mathcal{D}_k$`()` would return a set of descriptors, which generic element is $\mathcal{D}_b$.

The built descriptor, *i.e.* $\mathcal{D}_b$, would always have a ground equivalent to the type of the element in the entity set of $\mathcal{D}_k$, which is always equal also to the ground of $\bar{\mathcal{D}}$. In other words, we use an entity y_i as the ground of a new descriptor $\mathcal{D}_b$, which is instantiated with the ontology $\mathcal{O}$. $\mathcal{D}_b$ would have some sub-descriptors, which are defined by $\mathcal{D}_k$ in order to represent some axioms for the new ground, *i.e.* $\mathcal{D}_b$ describes some specific expressions $\mathcal{E}$. In this way, a new object $\mathcal{D}_b$ can be instantiated for setting and writing, or reading and getting, axioms in/to the ontology without relying on the factory but OOP paradigms.

Remarkably, for building complex entities as `OWLOOPRestriction` and `OWLOOPLink` we the grounding of a new descriptor is an issue, because the internal state it would have is not directly readable or writable in an ontology. For instance, we assume that the building operation when $y_i = \langle \mathbf{P}, \beta \rangle_i$ returns descriptors grounded on β_i, *i.e.* we do not build any axioms specifically on $\mathcal{P}_i$. However, in the case in which $y_i = \langle q, \mathbf{P}, \Delta \rangle_i$, it is not clear which should be the ground of the descriptor to be built, and at the current implementation, we do not provide any building support for `OWLOOPRestriction`. Nevertheless, since the modularity of OWLOOP, we argue for systems that use composition of descriptors, rather than implementing complex and not always intuitive synchronisation procedures with complex grounds.

5.8 OWLOOP API Examples

This section presents a toy example to show typical design patterns for using OWLOOP. In this example, we consider the ontology $\mathcal{O}$ as in Figure 5.1, where the individual `Robot1` is supposed to occur in axioms like `isIn(Robot1,Corridor1)`, where $\forall$`isIn.LOCATION` and `ROBOT: Robot1`. We want to notify the robot when some external signal occurs, and when this happens, it should reason on the *linked* locations it can reach.

We assume the robot to know the name of the individual describing its current location, *e.g.* `Corridor1`, and as output, we expect a list containing the reachable places express in terms of an individual and its type. As presented in an example of Section 5.6, also in this section we assume the type of an individual to be the deepest node in the class hierarchy it is an instance of, *e.g.* `Room1` is of type `ROOM`, while $\top$ and `LOCATION` should not be considered even if they are consistent classification of such an individual.

Listing 5.1 defines a pseudo-code for update the representation given an external signal coming from a bigger architecture that controls the robot. It presents a loop to check incoming signal frequently, or with a publish/subscribe paradigm for instance. At line 5.1.2, *pose* is retrieved as the actual position of the robot expressed as the relative IRI of an individual in the ontology. At Line 5.1.3, a new instance of the `TypeLinkIndividual` descriptor is created for representing axioms in $\mathcal{O}$ grounded in `Robot1`. Since the ground is an individual, this descriptor can contain only the sub-descriptors derived from Table 5.2. In particular, it involves $\mathcal{D}_{23}$ to represent the type of the ground and $\mathcal{D}_{24}$ to describe its properties. At Line 5.1.4, we set the internal state of $\mathcal{D}_{23}$, for representing that `Robot1` is an individual of the `ROBOT` class, and at Line 5.1.5 we set the axioms that specify the location of the robot. At Line 5.1.6, we write the internal state of the description for applying those axioms in the ontology. Then, we update the reasoner for – eventually – infer further knowledge.

Listing 5.2 reasons on the knowledge generated by Listing 5.1 in $\mathcal{O}$. At Line 5.2.1, we instantiates a new descriptor which represents

Listing 5.1: An example of semantic writing with OWLOOP.

```
    Descriptor : d, describes the location of Robot1.
5.1.1 while condition do
5.1.2     pose ← getNewPose()
5.1.3     d ← new TypeLinkIndividual(O, ‘‘Robot1’’)
5.1.4     d.setType(‘‘ROBOT’’)
5.1.5     d.setLink(‘‘isIn’’, pose)
5.1.6     d.write()
5.1.7     synchroniseReasoner(O)
```

only the $\mathcal{D}_{24}$ semantic, *i.e.* a `LinkIndividual`, involving the `Robot1` ground and its possible properties. At Line 5.2.2, we set the descriptor to synchronise the `isIn` property, and at Line 5.2.3 we read from the ontology. We assume in $\mathcal{O}$ that the robot can have only one position which, in accordance with the previous example, is the `Corridor1` individual. At line 5.2.4, we build a new description p based on such an individual, *i.e.* we obtain a set of descriptors R containing a single element that represents some axioms grounded on `Corridor1`. We define the sub-descriptor $\mathcal{D}_{24}$ of `LinkIndividual` such to return a set of `TypeIndividual` descriptors during building. The latter is a descriptor containing $\mathcal{D}_{23}$, which would have an individual as ground, and an entity set made of classes.

Between lines 5.2.5 and 5.2.7 we synchronise the internal state of p in order to build the locations linked to the current position of the robot. Therefore, at Line 5.2.8, we iterate over a set of `TypeIndividual` grounded first on `Room1` and than on `Room2`. For each of those individuals, at Line 5.2.9, we read from the ontology the classes in which they are an instance of, and we store them in the internal state of the t descriptors. Thus, at Line 5.2.10, we build such a description and we obtain a set of descriptions grounded on the classes having `Room1` or `Room2` as their instances, *i.e.* $\{\top, \texttt{LOCATION}, \texttt{ROOM}\}$ in both cases.

Listing 5.2: An example of semantic reading and building with OWLOOP. Given an ontology $\mathcal{O}$ as in Figure 5.1, this listing returns $\{\langle \texttt{Room1}, \texttt{ROOM}\rangle, \langle \texttt{Room2}, \texttt{ROOM}\rangle\}$.

Descriptor : d, describes the location of `Robot1`.
p, describes location liked to `Robot1`.
r, describes all the types of locations linked to `Robot1`.
t, describes types that subsume the locations linked to `Robot1`.

```
5.2.1  d ← new LinkIndividual(O, "Robot1")
5.2.2  d.setLink("isIn")
5.2.3  d.read()
5.2.4  p ← d.buildLinks()[0]                    // 1 LinkIndividual with x=Corridor1
5.2.5  p.setLink("isLinkedTo")
5.2.6  p.read()
5.2.7  R ← p.buildLinks()                       // 2 TypeIndividual, with, x={Room1,Room2}
5.2.8  foreach r ∈ R do
5.2.9      r.read()
5.2.10     T ← r.buildTypes()                   // 3 SubClass, both with x={⊤,LOCATION,ROOM}
5.2.11     foreach t ∈ T do
5.2.12         t.read()
5.2.13         W ← t.getSubTypes()              // all Λ_i s.t. Δ ⊏ Λ_i, where Δ is the ground x
5.2.14         if W.isEmpty() then
5.2.15             U.add(⟨r.ground(), t.ground()⟩)        // e.g. ⟨Room1,ROOM⟩
5.2.16             break
5.2.17 return U
```

Since we defined the sub-descriptor $\mathcal{D}_{23}$ of `TypeIndividual` such to build a set of `SubClass` descriptors invovling $\mathcal{D}_{13}$, t is a descriptor that synchronises subsuming axioms in its internal state, and this is done at Line 5.2.12. For each type of locations linked to the position of the robot, we get the internal axioms about subsumed classes, at Line 5.2.13. In our example, since $\top$ subsumes all the classes the condition at Line 5.2.14 is not verified because the internal state contains four classes, and this also happens for `LOCATION` since it has one subclass. In contrary, the condition at Line 5.2.14 is verified for `ROOM`, since it does not have any sub-class.

In this case, the output returned by the Listing 5.2 is a structure $U = \{u_1, u_2\}$, where $u_1 = \langle \texttt{Room1}, \texttt{ROOM} \rangle$, and $u_1 = \langle \texttt{Room2}, \texttt{ROOM} \rangle$, *i.e.* the reachable places and their types. Remarkably, since we exploit OOP paradigms through descriptor building, we had to explicitly state only the IRI of `Robot1`, `isIn`, and `isLinkedTo`, without the need to generate further constants about the names of entities in the ontology. Moreover, we presented how building mechanism allows extending and interfacing descriptors between each other, through different descriptors including combination of sub-descriptors in an OOP manner.

5.9 Conclusions

We presented the OWLOOP API, a library that allows to design algorithm using knowledge contained in an OWL ontology in an OOP fashion. We motivated why such a feature is crucial for bringing OWL in real robotic applications due to the complex software architecture they demand. Nevertheless, an ontology contains data structures that do not follow OOP paradigms, and design reasoners that can maintain an OOP-like ontology without loose knowledge expressibility is a state of the art challenge.

In this paper, we presented a formalisation and a proof of concept implementation that allows accessing ontology in an OOP manner, but its core uses standard libraries. In other words, we presented a wrapper that embeds a middle layer where we describe a fragment of the ontology that a user can exploit based on OOP interfaces and extensions.

The middle layer is in charge of performing the synchronisation between OWLOOP descriptors and OWL axioms in an ontology. For implementing such a procedure, we had to formalise a general interface representing an abstract descriptor. We design this interface to be very general, such to allows the definition of a descriptor of descriptors. We implemented and presented the most common sub-descriptors, and we discuss how their combination can be used for defining OOP structures of OWL axioms with a specific semantic.

With the purpose to increase the performances, OWLOOP is designed with the idea that a developer should make a particular de-

scriptor for each semantic in the ontology he or she wants to use. It this way, OWLOOP can be used for limiting as much as possible the knowledge to be synchronised. Nevertheless, OWLOOP supports the developer with the definition of many different sub-descriptors that can be extended and interfaced. At the current state, the OWLOOP functionalities cover an exhaustive set of descriptors for the most common applications, but it does not wrap all the possible OWL axioms.

6

A ROS Multi-Ontology References Service: OWL Reasoners and Application Prototyping Issues

Abstract

This paper introduces a ROS Multi Ontology References (ARMOR) service, a general-purpose and scalable interface between robot architectures and OWL reasoners. ARMOR addresses synchronization and communication issues among heterogeneous and distributed software components. As a guiding scenario, we consider a prototyping approach for the use of symbolic reasoning in human-robot interaction applications.

software architecture for robotics · knowledge representation · reasoning · description logics

Authors

Luca Buoncompagni*, Alessio Capitanelli* and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

*Published in the Proceedings of the 5th Italian Workshop on Artificial Intelligence and Robotics, a workshop of the XVII International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018, Trento, IT),*

and presented at the Autonomous Robot Ontology Workshop (ROMAN 2017, Lisbon, ES).

*These authors contributed equally to this work.

6.1 Introduction

The challenge of sharing and communicating information is crucial in complex human-robot interaction (HRI) scenarios. Ontologies and symbolic reasoning are the state of the art approach for a natural representation of knowledge, especially within the Semantic Web domain, and it has been adopted to achieve high expressiveness [11]. Since symbolic reasoning is a high complexity problem, optimizing its performance requires a careful design of the knowledge *resolution*. Specifically, a robot architecture requires the integration of several components implementing different behaviors and generating a series of beliefs. Most of the components are expected to access, manipulate, and reason upon a run-time generated representation of knowledge grounding robot behaviors and perceptions through formal axioms, with soft real time requirements.

The Robot Operating System (ROS) is a *de facto* standard for robot software development, which allows for modular and scalable robot architecture designs. Currently, some approaches exist to integrate a semantic representation in ROS, such as the KnowRob [155] which provide a complete framework of relevant ontologies, or the native ROS support of MongoDB[6.1], which can also be used to provide a suitable representation for semantic querying. Unfortunately, none of these supports the study of advanced reasoning paradigms, and they heavily rely on *ad hoc* reasoning solutions, significantly limiting their scope. We argue that this fact affects the study of different approaches to semantics in Robotics. For instance, it limits our capability to explore novel semantic representations of perceptions, which offers similar but *not equivalent* beliefs. We lack a standardized general framework to work with ontologies, natively supporting symbolic logic and advanced reasoning paradigms.

6.1 https://www.mongodb.com.

The Ontology Web Language (OWL)[6.2] is a standard representation supporting several reasoning interfaces, *e.g.* Pellet [144], and logic formalisms, *e.g.* the Allen's Algebra [5]. Thus, it can be a solid foundation for a framework for symbolic reasoning in Robotics. OWL is based on the separation between terminological and assertional knowledge, referred to as different *boxes* (Tbox and Abox). Typically, in Robotics scenarios, we design a *static* semantics for the beliefs to be represented in the TBox. Then, we populate the ABox through *individuals* defined using *types* and *properties* and, at run-time, we classify knowledge using *instance checking*. We argue that, due to the high complexity of HRI scenarios, the possibility of a dynamic semantics in the TBox is desirable as well. For instance, it could be used to *learn* new types for classification. This leads us to a study requiring reasoning heuristics to be compared, components to be shared, and different semantics to be adapted.

6.2 https://www.w3.org/TR/owl-guide.

For this purpose, we propose the ROS Multi Ontology References[6.3]

6.3 https://github.com/EmaroLab/ARMOR.

(ARMOR). ARMOR is an open source *service* which manipulates and queries multiple OWL ontologies. It provides access to a set of dynamic ontologies, handling also the synchronizations among different components in the architecture. Therefore, it is a convenient tool for managing knowledge representation supported by advanced reasoners.

6.2 System's Architecture

Figure 6.1 shows a schematic representation of ARMOR. It interfaces the OWL API [73] and reasoners through the Java-based Multi Ontology References library (AMOR). Then, ARMOR exposes AMOR functionalities as a service to ROS-based architectures, relying on the support for Java in ROS (ROSJava[6.4]). ARMOR messages have been designed to accommodate all OWL functionalities. Nevertheless, we have implemented only an exhaustive subset of those features so far (*i.e.* only common *run-time* operations). Indeed, ontology managers are not distributed across satellite components of a ROS architecture. Instead, dedicated components are in charge of management, while others only provide knowledge axioms, possibly at run-time. With ARMOR, it is possible to *inject* in the service procedure managing symbols from a centralized perspective, based on the functionalities provided by AMOR. Nevertheless, complex static representations can always be defined also off-line with dedicated software, *e.g.* Protégé[6.5].

6.4 https://github.com/rosjava.

6.5 http://protege.stanford.edu.

6.2.1 The ARMOR Core: AMOR

The core library, referred to as AMOR, contains a *map* of instantiated ontologies, where the *key* is a unique name identifier used to access the corresponding ontology in a thread-safe manner. AMOR provides several common operations, such as those to create individuals, assign their properties, make them disjointed and classify them, to name a few. Furthermore, AMOR ensures complete accessibility to the OWL API[6.6] features for compatibility and extendability purposes. For example, AMOR allows for invoking reasoners by specifying their `OWLReasoner` *factory*, *i.e.* the unique package of its Java implementation, which assures compatibility with all OWL reasoners.

6.6 http://owlapi.sourceforge.net.

In the current implementation, we interface several properties that are useful to tune the AMOR behaviour, *e.g.* the buffering of the manipulations or a continuous reasoner update, using the standard ROS Parameter Server, as well as parameters for debugging purposes such as toggling a Graphical User Interface (GUI) for visualising ontology states on-line.

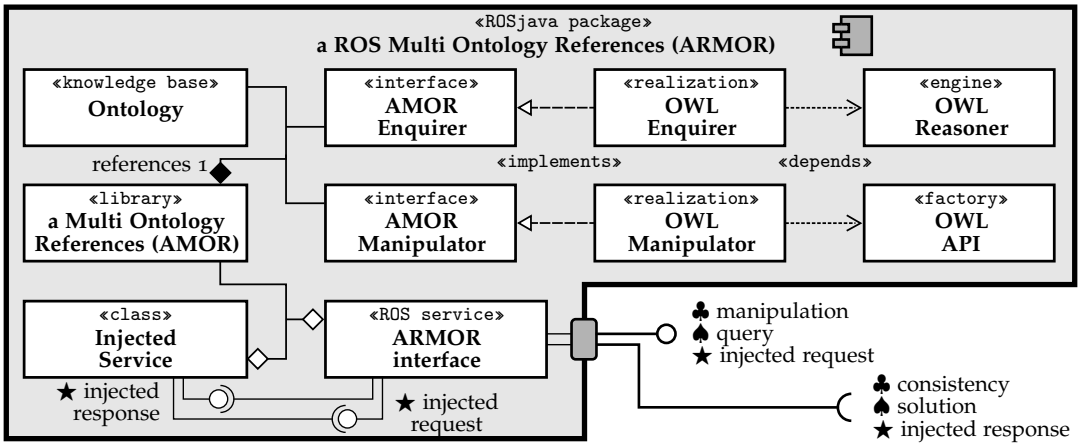


Figure 6.1: The UML diagram of ARMOR. It accepts two types of request and respond accordingly (♣, ♠). The request and response ★ are shown as an example where the Injected Service extends the services of ARMOR using AMOR.

6.2.2 The ARMOR Interface

The ARMOR interface is based on a ROS message structure (*i.e.* a triple) for the use of the AMOR functionalities from any node in the architecture, even when the development language is different from Java (*e.g.* Python[6.7] and C++ are the most common languages in Robotics development). Such a message is composed of:

6.7 An ARMOR client is available at https://github.com/EmaroLab/ARMOR_py_api.

- the *client name*, which is used by the service to identify different callers,
- the *reference name*, indicating the operation's target reference, and
- the *command* to execute, *i.e.* `add`, `remove`, `replace`, `query`, `load`, `mount`, etc.[6.8] Each of those commands may be further refined by:
  - the *primary* and *secondary specifiers*, which augment command labels, *e.g.* `add(`*`individual`*`, `*`class`*`)` or `remove(`*`individual`*`, `*`property`*`)`, and
  - the *arguments*, a list of entities in the reference parameterising the command, *e.g.* ⟨`add(`*`class`*`) "Sphere"`⟩, or even ⟨`add(`*`property`*`,`*`individual`*`) "hasNorth" "LivingRoom" "Corridor"`⟩.

6.8 Listed at https://github.com/EmaroLab/ARMOR/blob/master/commands.md.

An ARMOR call is based on one or more messages with the same structure. When such a request is sent, the service manipulates or queries the ontology with the given directives. Then, it returns whether the ontology is consistent, eventual error codes with their description, and the names of the queried entities, if requested. In other words, the interfaces ♣ and ♠ in Figure 6.1, and ★ can be defined for each specific injected service, if any.

One advanced feature of ARMOR is the possibility of flexibly synchronising all operations. This follows a *mounting/unmounting* paradigm, where one or more nodes identified by the same *client name* can prevent other nodes from manipulating a given ontology, in order to ensure manipulation consistency. On the contrary, queries are always allowed, except during reasoning time. Calls to busy ontologies will report a mounting issue, and the user can choose how to handle this situation.

6.3 Applications and Conclusions

We are currently using ARMOR in different applications, but here we mention only two of them. The first is aimed at implementing a dynamic PDDL problem generator. This approach uses descriptions of the *predicates* and *objects* in a tabletop scenario to infer unsatisfied norms and consequently generate goals [32]. The system has been integrated with ROSPlan[6.9] by substituting the internal semantic data structure with ARMOR and a suitable ontology.

The second application is a system to learn by demonstrations the arrangement of objects in the robot's workspace by mapping their properties into the TBox. In particular, we used an injected service[6.10] in ARMOR for performing scene learning and classification in a scenario where a robot explains its beliefs to a human, which might want to correct it through dialogues (in Chapter 9).

This paper introduces the ARMOR service to manipulate OWL ontologies and query their reasoners in a ROS-based architecture. ARMOR services are available through a flexible message allowing for the direct access OWL features from any component of the architecture. It ensures synchronisation between client calls and flexibility through procedure injection. Also, ARMOR allows for an easy interface between robotic architectures and OWL representations, and we practically showed it during a tutorial presented at the ROS Development Conference 2018[6.11]. The tutorial is focused on the control of mobile robots based on a topological environment representation and SLAM.

6.9 https://github.com/KCL-Planning/ROSPlan.

6.10 https://github.com/EmaroLab/injected_armor_pkgs.

6.11 http://www.theconstructsim.com/ros-developers-online-conference-2018-rdc-worldwide/ros-developers-conference-speaker-alessio-capitanelli. Code available at https://github.com/EmaroLab/armor_rds_tutorial.

Part IV

Demonstrate for Remembering

This part presents an algorithm we develop for creating and maintaining a graph of experiences in the robot's memory.

7

One-Shot Structured Learning of Scene Categories Through Demonstrations

Abstract

The paper presents an algorithm to learn a structure of scene categories with classifying purposes. The algorithm is based on symbolic representations, logic reasoning, and dynamic semantic manipulations. We deployed it in a ROS architecture through the definition of a semantic interface, which has been designed for parametrising the knowledge that the algorithm can process, *i.e.* the space in which it learns and structures experiences. We discuss a general-purpose implementation of the algorithm, and we deployed it in a simplified human-robot interaction scenario where tabletop scenes, made of objects spatially related, are (*i*) learned in a one-shot manner, (*ii*) structured based on sub-scenes previously experienced, (*iii*) classified with a similarity value, and (*iv*) explained to the user.

structured learning · robot memory ·
scene similarity representation · knowledge bootstrapping

Authors

Luca Buoncompagni and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

Paper to be peer-reviewed.

7.1 Introduction

Many approaches in Robotics foster a vision of healthcare where assistive robots take care of many aspects of our everyday life. Robots are expected to behave as proactive butlers, smart companions or social agents. Assistive robots will have to *understand* and adapt to the peculiarities of the environment (*i.e.* most likely private apartments) in which they will be deployed: not only will robots have to learn maps (possibly semantically), they will also have to locate and arrange objects (*e.g.* chairs, cups, glasses, plates, cutlery, books, etc.) in the way the assisted person is used to. At the same time, the robot should be aware of the person's activities, and it should contextualise its experiences in a structure that is suitable for specific tasks. Furthermore, preliminary evidence suggests the existence of an *interaction gap*, which relates the understandability among agents to the quality of the interaction [81]. Therefore, for having an efficient assistive robot, the user should be able to understand its experiences and intentions as well.

It is expected that an initial *knowledge bootstrap*, *i.e.* an understanding of the environment, will be conducted by the assisted person (or their relatives or caring staff) using familiar, language-based, qualitative concepts, such as *here in the kitchen*, *on the right-hand side of the dish plate*, or *beyond the water bottle*. We envision a scenario where anyone can teach a robot how to interpret scenes based on some semantic (*e.g.* spatial arranged objects), by showing and interacting. After the bootstrap phase, the robot should be able to maintain and retrieve the experiences it had based on the state of the environment. Once such capabilities are obtained, an assistive robot will be able to use learned representation for a variety of tasks, including communication purposes (*e.g. the cup is at the right-hand side of the plate*), action planning (*e.g. swap knife and fork*) or other tasks involving scene classification (*e.g. this scene corresponds to a sumptuously decked table*).

In particular, we want robots to (*i*) learn the configuration of objects through human demonstrations (*e.g. teaching by showing*, similarly to what has been described in [122, 123]), (*ii*) structure the semantic peculiarities of the scenes among subsequent demonstrations processes, (*iii*) classify the occurrence of a configuration that is *similar* to experiences it had, and (*iv*) communicate the classified experiences and their structure to the user. Since those operations are depending on the knowledge representation that the robot is supposed to build at runtime, the problem of design such a dynamic structure is crucial for implementing robots with these functionalities.

In practice, the design of the knowledge representation for robots is driven by the requirements of the application. Usually, knowledge representation is based on complex prior models that are assessed with respect to the state of the environment at runtime, and this is

used for *grounding* the intentions and actions of the robot. We argue for robots that aggregate prior knowledge to build models derived from experience, and we want to investigate a formalism to define prior knowledge that can be used with such a purpose. We want to use that formalism as an interface for the algorithm, which is made at the semantic level. In this way, the algorithm can be used in different applications based on appropriate prior knowledge, which will be exploited for learning and classifying structures of experience represented with a suitable semantic.

Our algorithm is based on the Description Logic (DL) formalism [11], and its operations are meant for maintaining a structure of experience encoded using the Ontology Web Language (OWL) [108]. We exploit OWL reasoners to organise the experiences in a hierarchy of classes which contains the representation of the scene to be recognised. The algorithm uses an ontology to process input facts about the scene, which generates beliefs that are matched with categories based on experience. As output, the algorithm returns a structure of categories that model some parts of the scene. In the output structure, categories are related among each other with a similarity value, which indicates how well their model describes the environment.

In this paper, we deploy the algorithm in a ROS architecture designed for a scenario aimed to highlight the behaviour of the algorithm. We consider a scenario involving a user and a robot facing each other at the opposite sides of a table where objects are spatially located, as shown in Figure 7.1. We detail the computation that the algorithm performs for some demonstrations in this scenario, and we report its outputs. We show the behaviour and the limitations of our algorithm through examples, which are aimed at presenting more general use cases where our method is effective.

The paper introduces related work, it gives an overview of the algorithm (in Section 7.3), and it formalises our problem from a set theory perspective (in Section 7.4) with the purpose to identify the operations to be performed by the algorithm as well as its interface. Then, the paper introduces the DL formalism as far as it is required for designing the algorithm, which is presented through a general-purpose implementation in Section 7.5. Section 7.6 details the semantic interface of the algorithm in a ROS architecture designed to evaluate scenes with spatial arranged objects. We observe the behaviour of the algorithm in such a scenario, and Section 7.7 reports the computation that has been performed for some examples. Finally, in Section 7.8 we discuss our algorithm, and conclusions follow.

7.2 Related Work

Recently, the problem of representing a spatial scene using geometric or probabilistic primitives gained significant attention in the literature.

A strand of approaches is focused on scene representation and understanding [157]. The need for topological, geometry-invariant and qualitative scene representations has been advocated in [34, 40]. An approach using relative spatial arrangements (regarding bounding boxes and centroids) among detected objects is presented in [64, 65] to improve the classification of 2D images. Relationships between different frames in video streams are learned in [47]. Specifically, correlations between regions with similar motions are detected. Joint distributions of spatial relationships over scenes is computed in [84], to recognise partially occluded objects using statistical information. Different spatial relationships are considered, namely bounding boxes of 3D objects, Euclidean distances and relative displacements. Scene similarity measurements for indoor environments are proposed in [3], where the authors model scenes (both individual objects and their relationships) using Gaussian Mixtures.

A second strand aims at representing scenes for manipulation purposes. Key structural visual features (and their relationships) are used in [103] as a representation framework to ground towel folding manipulation sequences. Proximity and contact relationships among objects from RGB-D data are determined in [129], where the scene is described by qualitative geometrical relationships in the form of a labelled graph. Labels represent the semantics associated with those relationships, *i.e. on top of* and *adjacent*. 3D vision histograms are used to learn manipulation-oriented spatial relationships in [59]. Surface patches are extracted from 3D point clouds, and histograms between patches are used to determine the occurrence of three relationships, namely *on top of*, *inside* and *rakable*. Finally, a mapping between object geometrical parameters and their representation is learned in [46] to represent information for manipulation tasks.

Many approaches to spatial reasoning have been proposed in the literature when the relationship in a scene are represented symbolically, *e.g.* the Region Connection Calculus or the Ternary Point Configuration Calculus [21, 41, 111]. However, such approaches are aimed at *reasoning*, not at human-robot interaction or learning. In particular, learning a purely symbolical model is an open issue that started to be tackled in the literature through probabilistic logic programming as reviewed in [56]. On the other hand, symbolic models can answer even very complex spatial queries, but their results are not necessarily suitable for human-like descriptions of spatial knowledge. Nevertheless, ontological representation has been used with interacting purposes since the user can interpret the knowledge it contains, *e.g.* in [93].

A benefit given by symbolic representation is the possibility to interface the symbols with different semantic than share the same representative structure, without affecting the algorithm consistently. For instance, temporal relations based on Allen's algebra [5] have been used for aggregate atomic activities in a smart environment (*e.g. go*

in the kitchen) in more complex models meant to recognise human activities [83]. That paper presents an ontology-based algorithm that is modular since it interfaces knowledge at the semantic level, which might encompass the different type of sensors and activity recognition modules, *i.e.* different knowledge representation sharing a common formalism.

Unfortunately, most expressive symbolic representations cannot deal with uncertain knowledge, and this requires to use perception techniques not affected by noise. Probabilistic [172] and fuzzy [24] reasoner for OWL ontologies are becoming mature, but unfortunately, their expressiveness might be still limited. Fortunately, OWL is a W3C standard that we used for prototype our algorithm, but we expect that more expressive reasoners will adhere to such language.

As far as the learning of structured symbols is concerned, a broadly investigated problem is to learn a structure of labels that characterise part of the scene, given as an image. For instance, in [96] reinforcement learning has been used for structuring labels segmented from an image in a graph representing actions, such as *holding* and *wearing*. In [147] a parser of images has been proposed to exploit knowledge extracted from an image through a recurrent neural network, which is structured in sentences based on recursive patterns, that explains the scene. In [164] and end-to-end learning technique is used for structuring the expression of a person, by considering its separated parts, *i.e.* eyes and mouth.

We are not aware of a system that learns a structured representation of scenes over time. Indeed, all the above methods structure data within the same scene, but not among scenes experienced in different situations. Usually, this problem is addressed with tracking techniques that are meant to maintain the representation of the environment that can be perceived. In a long-term scenario, this could be not suitable since the robot cannot perceive all the objects it should interact with, for all the time. If a tracking system is not used, the anchoring problem [42] is a known issue occurring when symbols that represented an object in a previously experienced scene should be matched when the object is observed again in the current scene.

7.2.1 Contribution

We propose an algorithm that exploits a purely symbolic representation in order to (*i*) represent qualitative scenes and reason on the semantic of their features, (*ii*) develop general-purpose implementation, and (*iii*) reduce the interaction gap with the user. To perform those tasks, we pair the algorithm with an OWL ontology, based on the DL formalism. We adopt DL since it is an expressive and computable standard, which also provides reasoners as Pellet [144], as well as

language for querying knowledge [143], and inferring methods, *e.g.* based on the Semantic Web Rules Language (SWRL) [74].

In particular, we address the *ii* point above through an interface that assures knowledge in the ontology to be available in a conventional format, that we define. The ontology is not limited to contains knowledge only with such a format, but the algorithm structures experience by processing only knowledge represented with our format. In other words, the algorithm is interfaced with the ontology through a set of known symbols that are defined in the semantic interface, and are supposed to be the results of some reasoning tasks, *i.e.* the inferred knowledge that features the scene at runtime. We design this interface based on a common technique, which represents the environment through a set of relations between pairs of elements (*e.g.* objects). Our algorithm does not fix the semantic of relations and elements to be considered in a scene, which are merely reduced to symbols. The combination of those symbols defines how the algorithm categorises experiences, as summarised in Figure 7.2.

The paper presents an algorithm that, given input facts describing the scene, returns categories ranked with a similarity value as output. The output is a structure containing all the previous experiences that match the environment, *i.e.* sub-scenes, which model are verified in the environment. During the learning phase of our algorithm, we learn scene categories through the observation of input facts representing the current scene. The algorithm implements one-shot learning, and we exploit it every time a scene cannot be classified (*i.e.* not previously experienced). Therefore, the algorithm always returns a classification of the scene which, if concerning a new experience, is leaned on-fly.

We present the algorithm in a scenario where objects are spatially related, and we detail the computation that is used for structuring experiences in a graph through instance and consistency checking performed by the OWL reasoner. Although our algorithm does not strictly require supervision, but in our scenario we involved a user in order to arrange the scenes that the robot should observe. Since we argue for an interaction that might help the robot, we consider the user as a potential supervisor, which monitors the robot perceptions, triggers the algorithm, and evaluates its output.

7.3 Overview

This paper addresses scenarios where the robot observes compositions of *facts*, *i.e.* *scenes*. The robot is supposed to learn scene *categories* that are classified based on similar compositions observed over time. Therefore, the robot maintains a structure containing categories modelling past scenes, which are used for classifying the current scene at runtime. For instance, it should represent that a *working station* is a scene

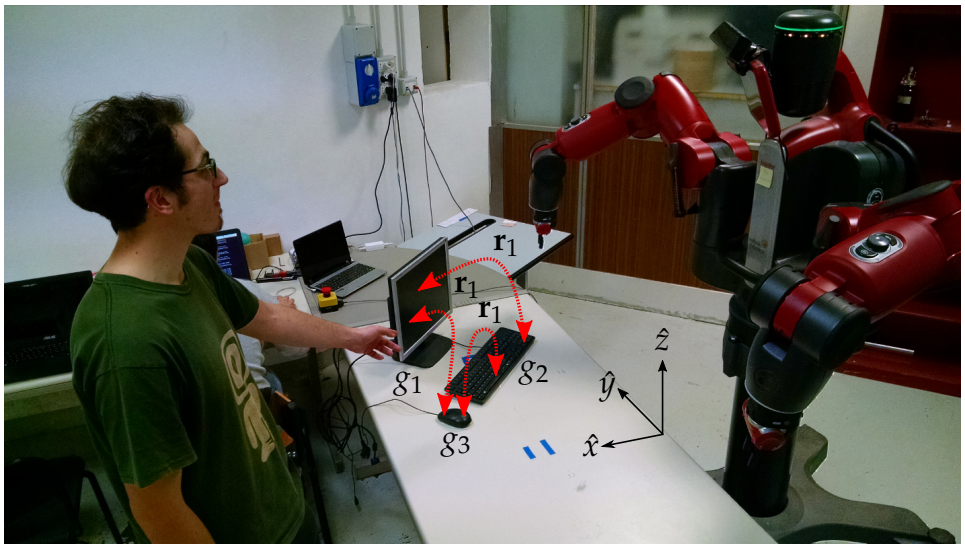


Figure 7.1: The reference scenario: a Baxter robot in front of a table where multiple objects (*i.e.* a monitor, a keyboard and a mouse) compose a scene shown by a human.

made of a monitor, a keyboard, a mouse, and the mutual relations that characterise them as shown in Figure 7.1.

We require the robot to reason on a symbolic (*i.e.* qualitative) description of the environment to simplify the interaction with a user, which can directly access and understand the robot's knowledge about the current and past scenes. For instance, symbolic facts of the example in Figure 7.1 can be based on a relation, which semantic is *to be connected to*, *i.e.* $\mathbf{r}_1$. Thus, in our representation, the objects g_i in the scene will be qualitatively related to each other. The algorithm uses an OWL ontology to reason over such a representation.

We aim to design a system that represents similarities among scenes by focusing their attention on particular *characteristics* of the environment (*e.g.* the colour, the shape, or the affordance of the objects in a table), which semantic is known a priory. We exploit symbolic representation to define general-purposes characteristics of scenes to be learned and classified, and we consider the *inclusion* as a measure of the similarity among scenes. In other words, we consider a scene to be included in another if the former is represented with all the facts modelling also the second, *i.e.* the latter is a more complex scene which includes a simpler sub-scene: the former category.

As an example, let extend the scene in Figure 7.1 with a chair and let us name the new category *office*. In this case, the *working station* category would be recognised in the new scene, since there are *at least* a monitor, a keyboard, and a mouse described through the relation $\mathbf{r}_1$. In particular, the algorithm would infer that an *office* includes a *working station*, and the similarity value between the two categories would be high since they differ in only one object (*i.e.* the chair).

The algorithm maintains a memory of scenes through three steps, *i.e.* the *perceiving*, the *classifying* and – eventually – the *learning* phases; all performed at each discrete instants of time. In the perception phase, we represent sensor data through symbolic *facts*, which are mapped into *beliefs* about a scene, as outlined in Figure 7.2. During the classification phase, we use beliefs for classifying the current scene with respect to previously learned categories. If no categories are recognised, the learning phase occurs for generating and structuring a new category from beliefs. Otherwise, all the recognised categories are returned by the algorithm with a *similarity value*, which measures their complexity based on current beliefs.

In the next section, we formalise our problem from a set theory perspective, where knowledge is encoded as elements in sets summarised in Figure 7.2. In particular, we define the perception, environmental and experience *spaces*, which are used to define the perceiving, learning, structuring and classifying *functions*. In Section 7.5 we define those spaces and functions with the DL formalism, and we present how OWL reasoning can be used for implementing the classify and structuring functions. For doing this, Section 7.4 defines the symbols that the algorithm requires in the ontology (*i.e.* its semantic interface).

7.4 Problem Statement

We represent a scene as a collection of the *facts* perceived at a certain instant of time, which contains n tuples f_i describing *instances*, $F = \{f_i, i \in [1 \dots n]\}$. Instances are particular elements of the environment that we want to take into account, *e.g.* they could be objects in a tabletop scenario, or activities in scheduling applications, as well as sub-symbolic quantities such as time instants or colours. Each tuple in F describes a fact with two instances g_x and g_y, which characterise the scene through a relationship $\mathbf{r}_z$, *i.e.* $f_i = \langle \mathbf{r}_z, g_x, g_y \rangle$. For instance, a fact could be a spatial relation in a tabletop scenario $f_1 = \langle \texttt{right,obj1,obj2} \rangle$, as well as an event $f_2 = \langle \texttt{alarm,room1,12:37} \rangle$, or a property $f_3 = \langle \texttt{color,cup2,white} \rangle$.

We define the *perception space* $\mathbb{F}$ spanning all the possible combination of observable facts. For a discrete instant of time, $\mathbb{F}$ contains all f_i characterising the environment through independent facts involving pairs of instances (g_x, g_y). At each instant of time, we want to represent the facts of the entire scene F as a single entity to reason with, *i.e.* we build a scene model. Therefore, we want to map all the facts in a unique symbol e, that we call *scene*. The scene is defined through a set of *beliefs* that are based on facts, and e is contained in the *environmental space* $\mathbb{E}$, which spans all the combination of beliefs that can be generated from a facts set in $\mathbb{F}$.

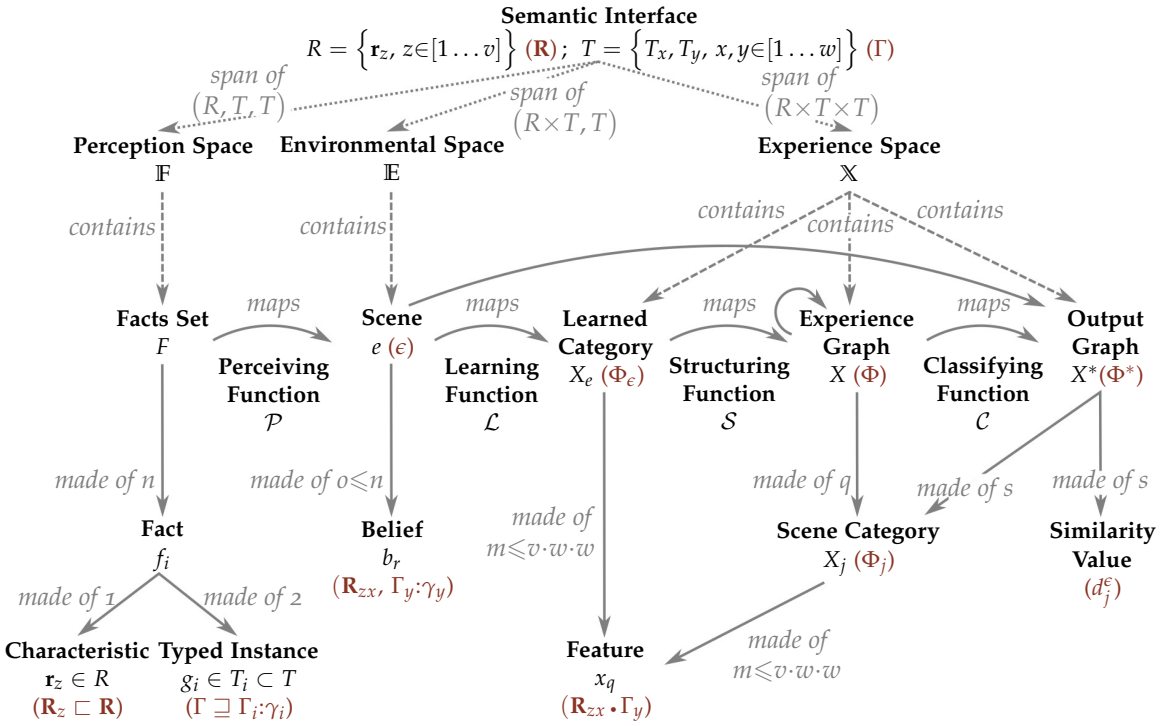


Figure 7.2: A glossary of the symbols (in black), and their relations (in grey), defined in this paper for representing the robot's knowledge. For a quick reference, we show in red some elements of the ontology that are relevant for implementing the related black symbols, which are introduced in Section 7.4 and motivated during the description of the algorithm in Section 7.5.

We define the *perceiving function* $\mathcal{P}$ as a map between a set of facts F and some beliefs b_r that define the components of the scene e, which is represented as a point in $\mathbb{E}$

$$\begin{aligned} \mathcal{P} : \mathbb{F} &\to \mathbb{E} \\ F &\mapsto e \doteq \{b_r,\, r \in [1\dots o]\}, \end{aligned} \tag{7.1}$$

where each belief b_r belongs to a space that *restricts* $\mathbb{E}$ based on the z-th *characteristic* and the x-th and y-th *instances* represented in a fact $f_i \in F$.

We want to divide $\mathbb{E}$ into *categories* (*i.e.* sub-spaces) that will be used for classifying the scene. In particular, we define the collection of all the scene categories as the *experience space* $\mathbb{X} \subset \mathbb{E}$. Formally, $\mathbb{X}$ contains a collection of sets X, where each generic set $X_j \subseteq X$ restricts the possible beliefs b_r that a scene should have to be classified, *i.e.* $e \in X_j$. If this occurs, we consider the scene to be *classified*. Otherwise, if $X_j = \emptyset$, we consider the scene e not being classified in the j-th category. If this occurs for all the categories, *i.e.* $e \notin X$, we want to learn a new category from the observations of the beliefs about e and store those in $\mathbb{X}$ for future classifications.

We structure the robot's experience X as an oriented graph of categories, *i.e.* an ordered set of sets $X \supseteq \{X_j,\, j \in [1 \dots m];\, \prec\}$ represented in $\mathbb{X}$, and defined through restrictions of beliefs in $\mathbb{E}$. We call X the *experience graph*, where each node is a scene category X_j, which is connected to another h-th category through an oriented edge *iff.* $X_j \prec X_h$. The *ordering operator* $\prec$ is a function that evaluates the *similarity* among categories, which are represented trough edges in X. In this way, the graph relates all the scene categories using a common method for measuring similarities among all the nodes (*e.g.* based on categories inclusions).

To generate categories in $\mathbb{X}$ from facts representing the scene $E = \mathcal{P}(F)$, we define the *learning function* $\mathcal{L}$ as a map between each belief b_r, to a *feature* x_q, which contributes to the definition of a new restriction of the environmental space X_e.

$$\begin{aligned} \mathcal{L} : \mathbb{E} &\rightarrow \mathbb{X} \\ e &\mapsto X_e \doteq \{x_q,\, q \in [1 \dots m]\}. \end{aligned} \tag{7.2}$$

We assume $\mathcal{L}$ to generate a new category X_e such to classify the scene e observed at learning time with a perfect similarity match. Formally, if a time t we compute $\mathcal{L}\left(e^t\right) = X_e$, then we classify $e^t \in X_e$ with the highest similarity value, *i.e.* there could not be any other categories X_j classifying a future scene e^{t+q}, with the same beliefs of e^t, better than X_e.

The learned category X_e in $\mathbb{X}$ is the definition of a new set that restricts a part of the possible beliefs in $\mathbb{E}$, but it is not a node in the experience graph yet. In order to include a new node X_e in the graph X, the system must update the graph's edges by computing the ordering operation $\prec$ among all the categories in the graph at the previous instant of time (X_j^{t-1}) and X_e. We perform this operation with the *structuring function*

$$\begin{aligned} \mathcal{S} : \mathbb{X}, \mathbb{X} &\rightarrow \mathbb{X} \\ \left(X^{t-1}, X_e\right) &\mapsto X, \end{aligned} \tag{7.3}$$

which computes a new experience graph X for the current instant of time given (*i*) the experiences graph at the previous instant X^{t-1}, and (*ii*) a new learned category X_e.

Given the experience graph in $\mathbb{X}$, we define the *classifying function* $\mathcal{C}$ as the operator that finds a sub-graph $X^* \supseteq \{X_k \subseteq X,\, \forall k \in [1 \dots s];\, \prec\}$, that we cal *output graph*. Given the current scene e and the experience graph at the current instant of time X we define X^* as

$$\begin{aligned} \mathcal{C} : \mathbb{X}, \mathbb{E} &\rightarrow \mathbb{X} \\ X, e &\mapsto X^*. \end{aligned} \tag{7.4}$$

We suppose that e belongs to all the categories $X_k \subseteq X^*$ by definition of $\mathcal{C}$, while the edges of X^* describe the similarity among each classified category X_k.

7.4.1 Dynamic Spaces Representation

As mentioned, the algorithm involves at each instant of time the computation of the *perceiving*, *classifying* and *learning* phases. The former involves the perceiving function $\mathcal{P}$, defined in Equation 7.1, while the last phase involves the learning function $\mathcal{L}$, in Equation 7.2. The classification phase relies on the classifying function $\mathcal{C}$ defined in Equation 7.4 and, if the learning phase occurs, $\mathcal{C}$ would be recomputed for updating the output of the algorithm. During the second evaluation of the classification phase, *i.e.* after any learning phases, the structuring function $\mathcal{S}$, defined in Equation 7.3, is evaluated before than $\mathcal{C}$.

More in particular, during the classification phase, we encode perceived facts in beliefs, and we generate the representation of the current scene e in $\mathbb{E}$. Based on restrictions of $\mathbb{E}$, represented in $\mathbb{X}$, we perform classification. If no categories are classified, the learning phase occurs for using e as a demonstration to generate a new scene category X_e. Then, we use the structuring function to arrange the new category in the experience graph, and we perform the classification phase once again to guarantee at least one categorisation of e for each instant of time.

We suppose that $\mathbb{X}^t \equiv X^{t-1}$ before the computation of any phases of the algorithm for each instant of time t, but during the execution of the algorithm this might not be true. Indeed, after the learning phase, $\mathbb{X}$ also contains a new category X_e^t, and when the structuring function has been computed $\mathbb{X}$ would contain a new graph X^t, which includes the new learned category in the experience graph represented at the previous instant. Thus, the algorithm maintains an experience graph X^t, which represents all the structured categories of scenes experienced from an initial instant of time, *i.e.* when the first facts have been processed. To simplify the notation, we implicitly consider that all the symbols involved in the algorithm are related to the current instant of time t. Otherwise, we explicitly indicate a temporal reference using right superscripts.

Figure 7.3a represents the perception space $\mathbb{F}$ which contains the facts at the considered time instant $(F^t, F^{t+1}, \ldots)$. At each instant of time, F contains some facts f_i expressed through tuples of the type $\langle \mathbf{r}_z, g_x, g_y \rangle$. Figures 7.3b and 7.3c represent both the environmental $\mathbb{E}$ and experience $\mathbb{X}$ spaces, but the first figure shows them during the learning phase at the time instant t, while the second figure depicts the representations during the classification phase at the time instant $t+1$. For each instant of time, facts F are mapped into a corresponding

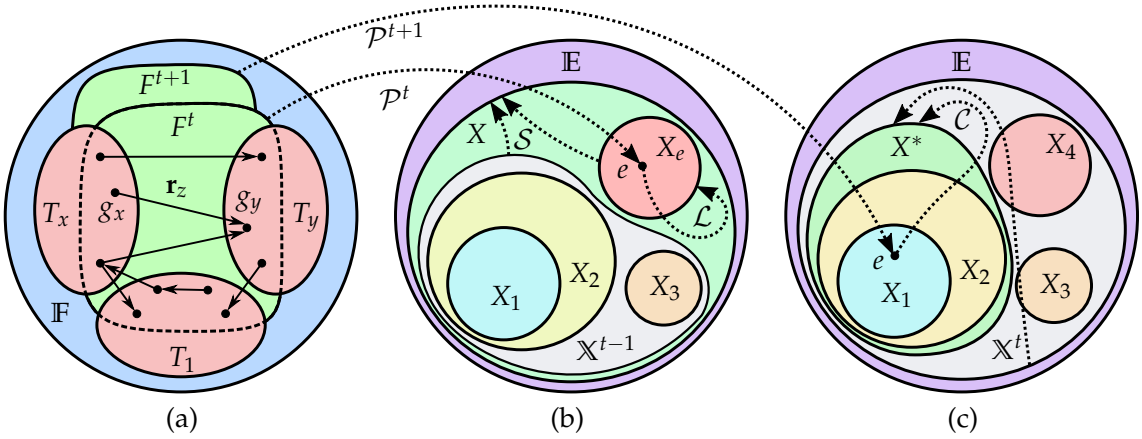


Figure 7.3: Evolution of the robot knowledge from a set theory perspective. Figure 7.3a shows the perceived facts F^t, F^{t+1} structured according to the proposed semantic interface at different time instants. Figures 7.3a and 7.3b show the robot knowledge in the learning phase at time t, where a new category X_e is derived from demonstration and included as a new experience. Figures 7.3a and 7.3c, instead, show the robot's knowledge during the classification phase of another scene e at time $t+1$, which is classified as X^*. Consequently, figures 7.3b and 7.3c represent the evolution of the experience space $\mathbb{X}$ over time.

representation of the current scene E in $\mathbb{E}$ by the perceiving function $\mathcal{P}$ that defines beliefs.

On the one hand, Figure 7.3b shows a case where the scene e generated from facts at time t does not belong to any categories known from previous experience X^{t-1}; therefore, it is necessary to learn it. Using the learning function $\mathcal{L}$, the robot gains a new scene category X_e, which can be grouped with the previous one by the mean of the structuring function $\mathcal{S}$. $\mathcal{S}$ computes a new graph X in $\mathbb{X}$ including $X_e \equiv X_4$ (notable from the comparison of figures 7.3b and 7.3c), which is returned as the only category classifying the scene, *i.e.* $X^* \equiv X_4$.

On the other hand, Figure 7.3c shows the situation when the scene e, associated with the fact F^{t+1}, is already contained in the previous robot experience X. Therefore, the robot can directly retrieve a subset of categories X^* by the mean of the classification function $\mathcal{C}$, without using $\mathcal{L}$. In this case, $\mathcal{C}$ returns a sub-graph $X^* \subseteq X$ having X_1 and X_2 as nodes (*i.e.* classifying categories), and edges such to identify if X_1, or X_2, is more similar to e.

7.4.2 Algorithm-Knowledge Interface

We design the algorithm to maintain the introduced representation over time with a general purpose set of symbols that follows a specific semantic interface. This interface represents the prior knowledge needed for encoding input facts and generating a consistent graph X in the experience space, which consequently identifies well posed classified categories X^*.

In particular, we defined each fact to be a tuple characterising two instances (g_x, g_y) through $\mathbf{r}_z$. We suppose the algorithm to known the possible symbols involved in facts, which are expressed in terms of (*i*) a *characteristic* spanning in a known set of relations $R = \{\mathbf{r}_z, \forall z \in [1 \dots w]\}$, and (*ii*) two *types* spanning in a known set of sets, $T \supseteq \{T_i, \forall i \in [1 \dots v]\}$, which is assigned to each instance, *i.e.* $g_x \in T_x$ and $g_y \in T_y$ with $T_x, T_y \subseteq T$. Also, we assume that types T_x and T_y describe each instance with a non-ambiguous type, *i.e.* $\bigcap_i T_{i=1}^{v} = \emptyset$; as shown in Figure 7.3a. Thanks to such an input structure we define $\mathscr{T}$ as the operator that returns a type given an instance, *i.e.* $\mathscr{T}(g_i) = T_i \subseteq T$.

Therefore, the semantic interface of the algorithm can be defined as the sets R and T used for generating a fact,

$$f_i = \langle \mathbf{r}_z, g_x, g_y \rangle \in F : \mathbb{F} \iff \mathbf{r_z} \in R,\ g_x \in T_x,\ g_y \in T_y, \tag{7.5}$$

which represents the definition of a fact that the algorithm can process and, consequently, the symbols that the ontology should contain in order to represent collections of perceived facts over time.

Through this interface, we assure that the algorithm outcomes X^* will contain categories based on characteristics R (*e.g.* *connected*) and types T (*e.g.* *monitor*, *keyboard* and *mouse*). Remarkably, this definition of the semantic interface of the algorithm is enough to derive all the dimensions spanning $\mathbb{F}$, $\mathbb{E}$ and $\mathbb{X}$, as we will discuss in Section 7.5.

Symbolic Representation

[...][7.1]

7.1 An OWL2-DL primer can be found in Chapter 5.

7.5 The Scene Identification and Tagging Algorithm (SIT)

In this section, we present an implementation of the phases of the algorithm based on the spaces and functions presented in Section 7.4. We suppose the algorithm to be supported from on OWL ontology and a related reasoner, and we will map all the symbols presented above in a DL formalism as summarised in Figure 7.2.

7.5.1 Perceiving

The perceiving function computes the scene as a point $e = \mathcal{P}(F)$ (Equation 7.1). This function is required for mapping the beliefs related to a single symbol e from all the facts F, which involve characteristics between independent pairs of instances (Equation 7.5), *e.g.* $\langle \texttt{connected}, g_x, g_y \rangle$, which might occur for each possible combinations of x, y spanning in $[1 \dots u]$, $x \neq y$, where u is the total amount of the

different instances occurring in all the facts F at time t, *e.g.* $u = 3$ in Figure 7.1.

As mentioned, our algorithm requires an ontology with prior knowledge describing the possible characteristics R and types T involved in facts. In particular, we define in the ontology each characteristic in R as a propriety $\mathbf{R}_z$ between individuals γ_x and γ_y, which are instances g_x, g_y. Moreover, types T are disjointed classes structured in a hierarchy having Γ as a root class, *i.e.* in the ontology all types T_i are considered as classes $\Gamma_i \sqsubseteq \Gamma$. For instance, figures 7.4a and 7.4c shows the structured prior knowledge in the ontology regarding the semantic interface that we used for the scenario considered in Section 7.6).

We exploit our semantic interface and specify the possible fact that the algorithm accepts by rewriting Equation 7.5 in the DL formalism as

$$\begin{aligned} f_i \equiv \left\{ \mathbf{R}_z \left(\gamma_x, \gamma_y\right),\ \Gamma_x : \gamma_x,\ \Gamma_y : \gamma_y \right\} \\ i \in [1 \dots n],\ z \in [1 \dots v],\ x, y \in [1 \dots w], \end{aligned} \tag{7.6}$$

where n is the number of facts perceived a the current instant of time.

For instance, a fact f_i in the scene in Figure 7.1 could be $\texttt{connected}(\gamma_x, \gamma_y)$, where it is known that $\texttt{KEYBOARD}:\gamma_x$ and $\texttt{MOUSE}:\gamma_y$. Where we assume to have prior knowledge in the ontology representing the `conncted` characteristics as a property in a known set of properties $\mathbf{R} \sqsubseteq \{\mathbf{R}_z, \forall z \in [1 \dots w]\}$, and types `KEYBOARD` and `MOUSE` as disjointed classes in $\Gamma \sqsubseteq \{\Gamma_i, \forall i \in [1 \dots v]\}$.

The perceiving function $\mathcal{P}$ is used for representing in the ontology the scene through a set of beliefs e. In particular, we define $\mathcal{P}$ such to map each fact f_i in a different belief b_r, which is represented in the ontology as a property that relates ϵ, `connected`, `KEYBOARD` and `MOUSE` consistently, for instance. In formula,

$$\begin{aligned} \mathcal{P}\left(F\right) =& \mathcal{P}\left(f_i\right)\ \ \forall i \in [1 \dots n], \\ \mathcal{P}\left(f_i\right) &= \mathscr{R}\big(\mathbf{R}_z, \mathscr{T}\left(\gamma_x\right)\big)\Big(\epsilon,\ \gamma_y\Big) \qquad\qquad z, x, y : f_i \\ &= \mathscr{R}\big(\mathbf{R}_z, \Gamma_x\big)\Big(\epsilon,\ \gamma_y\Big) = \mathbf{R}_{zx}\Big(\epsilon,\ \gamma_y\Big) \doteq b_r \end{aligned} \tag{7.7}$$

where $\mathscr{R}$ is a *reification* function based on our semantic interface that appends the name of a characteristic $\mathscr{N}(\mathbf{R}_z)$ with the name of the type of the instance involved in the domain of a fact $\mathscr{N}(\Gamma_x)$, *i.e.* $\mathscr{T}\left(\gamma_x\right) = \Gamma_x$ and $\mathscr{N}(\Gamma_x) = \texttt{KEYBOARD}$.

For instance, $\mathscr{R}\left(\mathbf{R}_z\Gamma_x\right) = \mathbf{R}_{zx} = \texttt{connectedKEYBOARD}$, consequentially a belief b_r could be for instance $\mathcal{P}(f_i) = \texttt{connectedKEYBOARD}(\epsilon, \gamma_y)$. In other words "the scene ϵ has a `KEYBOARD` `connected` to γ_y". Noteworthy, the ontology contains also $\Gamma_y \equiv \texttt{MOUSE}:\gamma_y$, which is known from Equation 7.6.

We suppose e to have o different beliefs b_r, based on the n facts given at the current instant of time. We observe that Equation 7.7 identify

two equivalent beliefs from two facts *iff.* they share the same (*i*) z-th characteristic, and (*ii*) type Γ_x, and (*iii*) instance γ_y. When those three conditions occur, the resulting beliefs from Equation 7.7 are indistinguishable, and we consider them only once in the definition of e. Therefore, at the same instant of time, the number of beliefs is always smaller or equal to the number of facts, *i.e.* $o \leqslant n$.

Remarkably, $\mathscr{R}$ spans a space among the $w{\cdot}v$ combinations of characteristics and types, *i.e.* the $R{\times}T$ space, which is represented in the ontology through the characteristics in $\mathbf{R}$ and on the types in Γ. Those combinations are supposed to be given as prior knowledge in the ontology and are identified by the semantic interface of the algorithm.

7.5.2 Learning

The learning function $\mathcal{L}$ creates categories in the ontology that classifies the beliefs about the current scene ϵ, which is represented as an ontological individual defined through properties computed from $\mathcal{P}$, in accordance with Equation 7.7. We define this function as

$$\begin{aligned}\mathcal{L}(\epsilon) &= \Phi \bigsqcap_{zx,y:b_r}^{r\in[1\ldots o]} \geqslant 1 \;\; \mathbf{R}_{zx}.\mathscr{T}(\gamma_y) \\ &= \Phi \bigsqcap_{zxy:x_i}^{i\in[1\ldots m]} \geqslant k_{zxy} \;\; \mathbf{R}_{zx}.\Gamma_y \doteq \Phi_\epsilon,\end{aligned} \tag{7.8}$$

which generates a class $\Phi_\epsilon \sqsubseteq \Phi$ that represents a new category X_e of the space $\mathbb{X}$ in the ontology. In Equation 7.8, we generate scene categories as a conjunctions of *features* $x_i \in X_e$, which are represented in the ontology as minimum cardinality restrictions ($k_{xyz} \in \mathbb{N}$) over the axioms $\mathbf{R}_{zx}.\Gamma_y$, for each different beliefs b_r.

It is important to notice that the restrictions spans in $m{=}w{\cdot}v{\cdot}v$ combinations of the $R{\times}T{\times}T$ space. The learning function $\mathcal{L}$ is design to count all the occurrences of a particular zxy-th combination defining a belief b_r. The counting value is store in k_{zxy}, and the sum of those value for all the m restrictions is equivalent to the number of beliefs o obtained at learning time. In other words Equation 7.8 generates o restrictions with k_{zxy} always equal to 1. Than, we aggregate all the m restrictions related to the same zxy combination by summing up all the restrictions. Since we adopted the open-word assumption, the restrictions that are not generated form beliefs, *i.e.* $k_{xyz}{=}0$, are not represented in the ontology. Therefore, the number of features of the learned categories Φ_ϵ could be generically less than m, and if $o < m$, than the number of features will be less or equal to than o.

Following the example in Figure 7.1, a feature of the Φ_ϵ category could be $\geqslant$1`connectedKEYBOARD.MOUSE`, identifying that ϵ is a scene of the Φ_ϵ category *iff.* at least one belief b_r involves a `connectedKEYBOARD`

and an instance of MOUSE. From the figure it is possible to deduce the other feature that is $\geqslant 1$connectedKEYBOARD.MONITOR In other words, the learned category can be read as "scene ϵ is of the category Φ_ϵ if at least one KEYBOARD is connected to a MOUSE, and if at least one KEYBOARD is connected to a MONITOR".

Remarkably, $\mathbb{E}$ is a theoretical space that represents all possible scenes in the environment, *i.e.* it is a huge space, while the size of $\mathbb{X}$ depends only on the robot's prior knowledge, and it contains a number of partitions that increases at each learning phase. At the beginning, we suppose $\mathbb{X} \equiv X^{t_0}{=}\emptyset$, and over time, $\mathbb{X}$ tends to be equivalent to a part of $\mathbb{E}$ that is observable with a semantic interface. This occurs thank the structuring function that includes new learned categories Φ_e in the experience graph.

7.5.3 Structuring

We defined the perceiving and learning functions in the DL formalism through equations 7.8 and 7.7 in order to exploit logic reasoning for performing the structuring and classifying functions. We focus on the former function in this section, while the latter is addressed in the next section.

Section 7.4 introduced the structuring function $\mathcal{S}$ as an operator that processes the ordering operator $\prec$ for all the pairs of categories in $\mathbb{X}$. This operation builds the edges of the experience graph, which represent categories similarities. Therefore each time a category Φ_ϵ is learned, $\mathcal{S}$ should include it as a new node in the experience graph represented at the previous instant of time, and update all the edges consistently before to re-performing the classification phase.

We identify the experience graph X as all the classes that are included ($\sqsubset$) in the Φ class in the ontology. Since Equation 7.8 defines each new category as a conjunction of Φ and some features, in the ontology the reasoner infers that $\Phi_\epsilon = \mathcal{L}(\mathcal{P}(F)) \sqsubseteq \Phi \ \forall F$ in $\mathbb{F}$, by definition of $\mathcal{L}$. In other words, Φ is the root of the experience X implemented in the ontology as an acyclic graph where we identify also some leaves, and each edge represents a logical implication between two scene categories $\Phi_j, \Phi_h \sqsubseteq \Phi$. In our implementation, the edges of the experience graph are automatically updated by the OWL reasoners that evaluates $X_j \prec X_h$ as $\Phi_j \sqsubset \Phi_h$, for each pairs of categories j, h experienced until the current instant of time.

If the ordering evaluation between the j-th and h-th category is verified, we consider the category Φ_h to be included in Φ_j (or Φ_j *implies* Φ_h), since all the features of Φ_h are also satisfied in the restrictions of Φ_j, *i.e.* $\Phi_j \Rightarrow \Phi_h$, which identifies the direction of the edge in the graph as shown in Figure 7.4b from the children to their parents.

Also we observe that a loop between two nodes in the graph represents equivalent categories that are simplified, *i.e.* $(\Phi_j \sqsubset \Phi_h) \wedge (\Phi_h \sqsubset$

$\Phi_j) \Rightarrow \Phi_h \equiv \Phi_j$. Moreover, in our ontology, a deeper of a node in the experience graph represents a more complex scene, which has common features (*i.e.* similarities) with their parent and assessors. In Section 7.5.5, we design the similarity value to quantify differences in categories complexity.

7.5.4 Classifying

Given the experience graph Φ evolving based on the learning and structuring functions, we want to classify a current scene against previously experienced categories with some similarities. In particular, Equation 7.4 defines the classification function $\mathcal{C}$ as an operator that finds all the categories in the knowledge representation having features matching a scene ϵ. In Equation 7.4 we consider the classification as a sub-graph Φ^* of the experience graph Φ; containing learned categories as nodes. In the ontology we represents the output graph $\Phi^* = \mathcal{C}(\Phi, \epsilon)$ as a hierarchy of classes containing s categories $\Phi_j \sqsubset \Phi$ all classifying the current scene, *i.e.* $\forall j \in [1 \dots s]$, $\Phi_j : e$ is consistent in the ontology.

Since we implement a perceiving function (Equation 7.7) that represent the scene as individual ϵ, and a learning function (Equation 7.8) as an operation that generates classes, we can exploit logic reasoning to query the classification of a scene at runtime. It is possible to express such query with an incomplete axiom formalised with an SWRL rule

$$\mathcal{C}(\Phi, \epsilon): \quad \Phi_? \sqsubseteq \Phi^* \iff (\Phi_? \sqsubseteq \Phi) \wedge (\Phi_? : \epsilon), \tag{7.9}$$

where ? denotes a variable considered for solving the query.

Through the query, we obtain the nodes of Φ^* as (*i*) all the classed in the ontology that implies Φ (*i.e.* ? can be only nodes Φ_j in the experience graph), which (*ii*) classify the current scene ϵ. We observe that the definition of $\mathcal{P}$ and $\mathcal{L}$ are such to assure point *ii* above *iff.* all the features of a category Φ_j have restriction matching some beliefs b_r about the scene. We consider a category feature to match a scene belief when they share the same reified relation $\mathbf{R}_{zx}$, which is applied between ϵ and γ_y, as well as in a restriction involving Γ_y.

The result of the query identifies the output graph X^* as a hierarchy of classes having Φ as root and the most complex categories of ϵ as leaves, *i.e.* categories with more restrictions. Remarkably, we define the query that also returns the root of the experience graph. Therefore, the output graph contains always Φ, and if it does not contain any further categories, the scene is considered not classified. For instance, consider ϵ to be represented by beliefs obtained from the first scene presented to the robot, which has an experience graph as shown in Figure 7.4b. In this case, the query will return an output graph Φ^* composed by the root, as well as the nodes `SCENE2` and `SCENE1`, which is the only leaf.

In particular, the leaves of the output graph Φ^* are the categories having a higher number of features matching the beliefs of ϵ, *i.e.* they are the most similar classified categories. Their parents are other simpler categories having all the features matching a lower number of beliefs, thus they categorise a smaller part of the scene which is in common with their children, *i.e.* a sub-scene. Their ancestors are yet more simple categorisations of ϵ, and the root categorises empty scenes.

7.5.5 Scene Similarity Measure

We compute a *similarity value* between the current scene ϵ and all the categories in $\Phi_j \sqsubset \Phi^*$ obtained from the classifying function, except for the root class Φ, which is considered to have no similarity with any scenes (*i.e.* its *similarity value* is always 0). In particular, we measure the similarity between the classified category as a confidence value representing the number of beliefs matched between ϵ and the features of each category in Φ^*. We assume that more the matchings, the more the similar the scene is with a category since it contains more restrictions that consistently represent beliefs. A perfect match among the beliefs of a scene and the features of a category is considered as a classification with the highest similarity value, supposed to be 1.

For defining the similarity value between a scene ϵ and a category $\Phi_j \sqsubseteq \Phi^*$, we compute the *scene cardinality* d_ϵ as the amount of different beliefs generated from the n facts

$$d_\epsilon = \sum_{f_i \in F}^{i \in [1 \dots n]} \mathscr{D}\Big(\mathcal{P}\left(f_i\right)\Big), \tag{7.10}$$

where $\mathscr{D}$ is a counting function such to returns 1 for each facts without counting duplicated beliefs involving the symbols $\mathbf{R}_{zx}$ and γ_y. in accordance with the definition of $\mathcal{P}$ in Section 7.5.1, d_ϵ is equal to the size of e, *i.e.* ${}_\epsilon d \doteq o \leqslant n$.

Furthermore, given a graph Φ^* composed by classes Φ_j having ϵ as one of their instances, we define the j-th *category cardinality* as

$${}^j d = \sum_{zx,y:\Phi_j}^{m} k_{zxy}, \tag{7.11}$$

which is equal to the amount of restrictions in Φ_j that are features matching some beliefs of the categorised scene, $\Phi_j : \epsilon$. Remarkably, ${}^j d$ is equivalent to the number of beliefs o that were in the ontology during the learning phase of the j-th category, which is obtained as the sum of all the restrictions k_{zxy}.

We compute the similarity value among classified scenes as the ration between the cardinalities defined in equations 7.10 and 7.11. In

particular, we assign to each j-th node in the output graph a value ${}^{j}d_{\epsilon}$ identifying the similarity between the scene ϵ and the j-th category as

$$ {}^{j}d_{\epsilon} = \frac{{}^{j}d}{d_{\epsilon}} \in [0,1] \subset \mathbb{R}, \tag{7.12} $$

which is a real number spanning from 1, representing the highest similarity (*i.e.* a perfect match between the j-th category and ϵ), to 0, the lowest similarity value. This is true since $d_{\epsilon} \geqslant {}^{j}d$, otherwise ϵ would not be an instance of the Φ_j category; therefore, Φ_j would not be considered in Equation 7.11 since $\Phi_j \not\sqsubset \Phi^*$.

Remarkably, ${}^{j}d_{\epsilon} = 1$ *iff.* (*i*) the scene category is the one learned in the same instant of time, or (*ii*) the j-th class categorises a scene that is equivalent from the once processed during learning phase at time t_j, *i.e.* the current scene and the scene ϵ^j are described with the same beliefs. This implies, that at each instant of time the algorithm always returns an output graph Φ^* containing some classification. If the classified categories have not been obtained from previous experience, the learning phase is used for generating a new category with similarity value equal to 1 by definition of d_j^{ϵ} (Figure 7.3b). Otherwise, if a classified category has been learned from previous experiences, Φ^* will contain some sub-spaces of $\mathbb{X}$ with equal or lower similarity values, as shown in Figure 7.3c.

Also, it is important to note, that for the definition of $\mathcal{L}$ in Equation 7.8, any learned sub-space of $\mathbb{X}$ does not overlap, or includes each other, in accordance with Figure 7.3b. In particular, out representation of the ordering function ($\prec$) do not support categories overlapping since it is implemented in the ontology as an implication ($\sqsubset$). Furthermore, with the presented implementation it does not also support learn categories that represent scene inclusion, *i.e.* it is not possible to learn a category Φ_{ϵ} included in another Φ_j^{t-p} learned at previous time instance. This happens because Φ_j^{t-p} would solve the query that is implemented by the classifying function, and the learning phase would not occur. Nonetheless, the output category Φ_j^{t-p} could have a low similarity value, denoting that the category matches only a few beliefs about the scene, but the robot would not create a more complex category for such a scene.

For solving this issue, we extend the classification function considering the similarity values. In particular, we rewrite the query in Equation 7.9 such to have solutions involving only classes implying the root of the experience graph and having ϵ as one of their instances as

$$ \mathcal{C}(\Phi, \epsilon, \ell): \quad \Phi_? \sqsubset \Phi^* \iff (\Phi_? \sqsubset \Phi) \wedge (\Phi_? : \epsilon) \wedge ({}^{?}d_{\epsilon} \geqslant \ell), \tag{7.13} $$

where we add the threshold ℓ such to consider as classified only the categories Φ_j having a minimum amount of features matching the

beliefs about ϵ, *i.e.* ℓ is a real number in $[0, 1]$ identifying the similarity value of a negligible category.

In Equation 7.13, we do not classify categories of ϵ having a low similarity value. Therefore the learning phase might occur when categories exist in the experience graph but do not have enough features that match the beliefs about ϵ and are not considered in the output graph. Remarkably, since the deep of the categories in the experience graph is related to their complexity, when a category is not considered since too simple, also all its parents will be disregarded. Therefore, to provide a consistent output, we can always build an edge from the most simpler classified categories to the root.

7.5.6 Algorithm Phases

Algorithm 7.1 shows the steps performed by our algorithm during the perceiving, learning, and classifying phases. The algorithm takes as input some facts F at an instant of time and returns an output graph Φ^* contained classified scene categories with similarity values valid for such an instant of time. The algorithm assumes to access an OWL ontology, which is stored as an internal *field*, and that contains the memory of the robot, as well as prior knowledge respecting the semantic interface.

Such knowledge is semantically expressed in the ontology through symbols, some of which are required for the algorithm. We define $\mathscr{N}$ to be the function that returns a symbol related to an element in the ontology, *i.e.* a IRI identifying its *name*. In particular, the algorithm assumes to find in the ontology symbols described as (*i*) the set of names of possible characteristic of a fact $\mathscr{N}(\mathbf{R})$, (*ii*) the set of names of possible types involved in a fact $\mathscr{N}(\mathbf{\Gamma})$, and (*iii*) the name of the root of the experience graph $\mathscr{N}(\Phi)$ (*e.g.* `Experience` in Figure 7.4b). Those values, given as prior knowledge in the ontology and constants in the algorithm are enough to implement the semantic interface.

The algorithm starts with the perceiving phase, which uses the semantic interface introduced in Section 7.4.2. Indeed, at Line 7.1.2, the algorithm retrieves the characteristic $\mathbf{R}_z$ and typed instances $\Gamma_x : \gamma_x$ and $\Gamma_y : \gamma_y$ from each input fact f_i, which is supposed to be represented consistently with the ontology, *i.e.* in accordance with the prior knowledge. For each $i \in [1 \dots n]$, where n is the number of input facts, the algorithm computes a scene belief that defines a component of the scene ϵ in $\mathbb{E}$. In particular, at Line 7.1.3, Equation 7.7 is computed iteratively, and all the beliefs are added to the ontology at Line 7.1.4.

At Line 7.1.5, we count the number of beliefs that represents the scene cardinality d_ϵ, in accordance with Equation 7.10. The perceiving phase ends, and the scene $\epsilon \in \mathbb{E}$ is generated in the ontology. The classification phase starts at Line 7.1.6, where the ontology is in charge to reason on the beliefs of ϵ.

Algorithm 7.1: The SIT algorithm. At each discrete instant of time, given new facts, it classifies, eventually learns, and returns a graph of scene categories based on previous experience.

Const : $\mathcal{N}(\mathbf{R})$ the name of fact's characteristics,
$\mathcal{N}(\Gamma)$ the name of fact's types,
$\mathcal{N}(\Phi)$ the name of the experience graph.

Field : the robot's knowledge in the ontology *onto*.

Input : facts at the current time instant $F \subset \mathbb{F}$.

Output : a grph of similar categories $\Phi^* \subset \mathbb{X}$.

// Perceiving Phase ($\mathcal{P}$)

7.1.1 **foreach** $f_i \in F$ **do** // $i \in [1 \dots n]$

7.1.2 $\quad \langle \mathbf{R}_z, \Gamma_x{:}\gamma_x, \Gamma_y{:}\gamma_y \rangle \leftarrow f_i$.semanticInterface()

7.1.3 $\quad \epsilon[i] \leftarrow \mathscr{R}(\mathbf{R}_z, \Gamma_x)\left(\epsilon, \gamma_y\right)$ // belief set

7.1.4 *onto*.add(ϵ) // Equation 7.7

7.1.5 $d_\epsilon \leftarrow \epsilon$.size() // Equation 7.10

// Classifying Phase ($\mathcal{C}, \mathcal{S}$)

7.1.6 *onto*.reason()

7.1.7 $\Phi^* \leftarrow$ *onto*.query($\Phi_? \sqsubset \Phi \wedge \Phi_? : \epsilon$) // Equation 7.9

7.1.8 **foreach** $\Phi_j \in \Phi^*$ **do** // $j \in [1 \dots s]$

7.1.9 $\quad$ **foreach** $\langle \mathbf{R}_{zx}, \Gamma_y \rangle \in \Phi_j$ **do** // $[1 \dots m]$

7.1.10 $\quad\quad k_{zxy} \leftarrow \Phi_j$.getFeatureRestriction($\mathbf{R}_{zx}, \Gamma_y$)

7.1.11 $\quad\quad {}^j d \leftarrow {}^j d + k_{zxy}$ // Equation 7.11

7.1.12 $\quad {}^j d_\epsilon \leftarrow {}^j d / d_\epsilon$ // Equation 7.12

7.1.13 $\quad$ **if** ${}^j d_\epsilon \geqslant l$ **then**

7.1.14 $\quad\quad \Phi_j$.setSimilValue(d_j^ϵ) // Equation 7.13

7.1.15 $\quad$ **else**

7.1.16 $\quad\quad \Phi^*$.removeNode(Φ_j)

7.1.17 **if** Φ^*.*nodesCnt()* > 1 **then** // $s > 1$

7.1.18 $\quad$ *onto*.remove(ϵ) // clear ϵ

7.1.19 $\quad$ **return** Φ^*

// Learning Phase ($\mathcal{L}$)

7.1.20 **else**

7.1.21 $\quad \Phi_\epsilon \leftarrow \Phi$

7.1.22 $\quad$ **foreach** $\langle \mathbf{R}_{zx}, \gamma_y \rangle \in \epsilon$ **do** // $[1 \dots o]$

7.1.23 $\quad\quad K[zx, \mathscr{T}(\gamma_y)] \leftarrow K[zx, \mathscr{T}(\gamma_y)] + 1$ // generate restrictions set

7.1.24 $\quad$ **foreach** $\langle \mathbf{R}_{zx}, \gamma_y \rangle \in \epsilon$ **do** // $[1 \dots o]$

7.1.25 $\quad\quad \Phi_\epsilon \leftarrow \Phi_\epsilon \sqcap \geqslant K[zx, \mathscr{T}(\gamma_y)]\ \mathbf{R}_{zx}.\mathscr{T}(\gamma_y)$ // generate new category feature

7.1.26 $\quad$ *onto*.add(Φ_ϵ) // Equation 7.8

7.1.27 $\quad$ **goto** 7.1.6 // it never reaches Line 7.1.21 again

In particular, at Line 7.1.7 we query the categories that are classifying the scene, and we obtained a graph of classes, which are not related to similarity values (Equation 7.9). For each category classified by the reasoner the algorithm computes the category cardinality ${}^{j}d$ (Equation 7.11, between lines 7.1.8 and 7.1.11), which is used at line 7.1.12 for identifying the similarity value between the scene ϵ and the j-th category, *i.e.* ${}^{j}d_{\epsilon}$ (Equation 7.12). If the similarity value is bigger than the threshold ℓ, we accept the classification, and we add to each node of the output graph the similarity value at Line 7.1.14. Otherwise, at Line 7.1.16, we discard the classification since the category is not similar enough to the current scene.

At Line 7.1.17 the algorithm checks if there is some classified categories, *i.e.* any other note in Φ^{*} apart from the root Φ. If this occurs, the algorithm returns an output graph Φ^{*} after having removed all the beliefs about the current scene. This is required for cleaning $\mathbb{E}$ by removing ϵ from the ontology, which at the next instant of time will depend only on the facts related to the instant of time $t+1$. Otherwise, at Line 7.1.20, the learning phase of the algorithm occurs.

At Line 7.1.21 the learned category Φ_{ϵ} is initialised to be equivalent to the root of the experience graph. In the loop at Line 7.1.22, the algorithm iterates over all the beliefs about the scene, and counts how many features $\mathbf{R}_{zx}$ relates ϵ to an instance of the same type Γ_{y}. After this operation we build a set of restrictions indexed by (zx, y), which is used at Line 7.1.25 for implementing the learning function $\mathcal{L}$ in accordance with Equation 7.8.

At Line 7.1.26, the algorithm adds the new category in $\mathbb{X}$, and then it goes to Line 7.1.6. Here the reasoner of the ontology is updated for performing the structuring function $\mathcal{S}$ before to classify ϵ in the categories of the experience graph as happened before. The classification phase is performed again, and by definition of $\mathcal{L}$, now Φ^{*} contains at least the learned categories Φ_{e} with a similarity value equal to 1; in accordance with Section 7.5.5. Therefore, the condition at Line 7.1.17 is always true after the learning phase, and the algorithm terminates by returning the output graph.

7.6 Implementation

To test Algorithm 7.1 in a tabletop scenario, we implemented a software architecture based on the Robotic Operative System (ROS). In this section, we present the design of a simple semantic interface that involves objects (*i.e.* instances), geometric shapes (*i.e.* types), and spatial relations (*i.e.* characteristics). In particular, we introduce a simple implementation that we used for discussing the examples presented in Section 7.7.

Between Line 7.1.7 and 7.1.16, the algorithm performs the classifying function $\mathcal{C}$ in accordance with Equation 7.13.

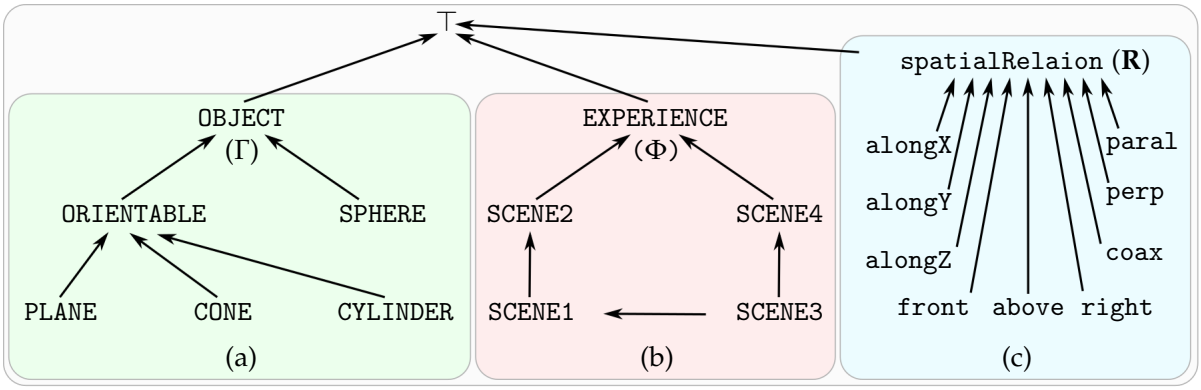


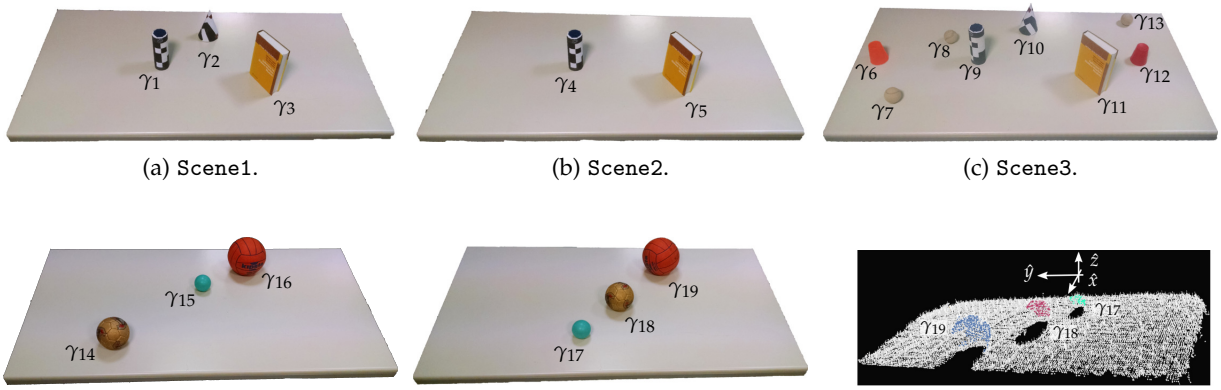
Figure 7.4: The classes and properties in the ontology where each arrow represents the inclusion operator, *i.e.* if $A \sqsubset B$ then the arrow starts from A (the children) and ends in B (the parent). Each fact is represented in the ontology with one characteristic of the hierarchy shown in (c) that relates two instances of the types shown as leaves in (a). Namely, (a) and (c) represent the semantic interface (R, T) in the spatial ontology used in the referencing scenario. Figure 7.4b shows the experience graph having categories of scenes as nodes, and edges indicating scenes inclusions. At the beginning we assume the experience graph Φ to do not have any children. At each learning phase, the algorithm generated a new category $\Phi_j \sqsubset \Phi$, and 7.4b shows the state of the robot's knowledge after that all the scenes in Figure 7.5 has been experienced.

7.6.1 Software Architecture and Knowledge Representation

Our architecture requires an ontology that we design with Protégé[7.2] in order to define the prior knowledge required by the semantic interface adopted in the referencing tabletop scenario. In the ontology, shown in Figure 7.4, tree mains disjointed hierarchies are semantically defined. On the one hand, to ground the semantic interface we used the $\mathcal{N}(\Gamma)$=`OBJECT` and $\mathcal{N}(\mathbf{R})$=`spatialRelation` descriptions, which are described in sections 7.6.3 and 7.6.4 respectively. On the other, the hierarchy of classes having as root $\mathcal{N}(\Phi)$=`EXPERIENCE` is our representation of the experience space $\mathbb{X}$, which is generated at runtime based on scene demonstrations through the learning and structuring functions. Over time, the algorithm changes the TBox of the ontology, which contains the experience graph classifying ϵ, while the ABox contains only individuals related to the current instant of time.

7.2 *https://protege.stanford.edu/*

Our algorithm is implemented in a ROS architecture where components can access ontology through the ROS Multi-Ontology References service (ARMOR) presented in Chapter 6. Based on the OWLOOP API (presented in Chapter 5), we extended the ARMOR services with an injected procedure that implements Algorithm 7.1 with the purposes of maintaining the experience graph in the ontology over demonstrations. The injected service we implemented accepts requests that trigger the algorithm assuming that the ontology contains facts that are retrieved in accordance with the semantic interface.



(a) Scene1. (b) Scene2. (c) Scene3.

(d) Scene4. (e) Scene5. (f) Segmented point cloud of Scene5.

Figure 7.5: Examples of scenes experienced at five different instants of time (figure from 7.5a to 7.5e). Figure 7.5f shows the perceived point cloud for the fifth scene, where each colour identifies a cluster of points assigned to an object. The spatial relations are expressed with respect to the robot's orientation, which faces the table from the reader perspective in the figures from 7.5a to 7.5e. In the last figure, we show the perspective of the user, which is aware of the static reference frame $\hat{x}\hat{y}\hat{z}$, shown in Figure 7.1.

Therefore, we assume that the architecture contains at least a perception module that uses ARMOR to generate facts in the ontology. We consider in the architecture also an auxiliary module that triggers the algorithm and presents the output to the user. Though this type of interface we made the algorithm depending on the perceiving modules at the semantic level. In this way, our implementation supports different combinations of perception techniques, as well as semantic interfaces, and usage of the output graph.

The paper discusses a referencing scenario where objects and spatial relations, as well as the human-robot interaction, are simplified because our objective is to focus on the structured learning of scene categories. We formalised the algorithm for general facts semantics, but the semantic interface adopted for a particular application affect the features of the categories that can be represented in the ontology. Although the problem of finding sets of characteristics and types to be interfaced with the algorithm for an expressive representation of scenes is out of the scope of this paper, in sections 7.7 and 7.8 we preliminary discuss some observation in our simplified scenario.

Nevertheless, more sophisticated perception and interacting techniques are indeed required for an architecture that uses our algorithm in a real scenario. The design of such techniques is strongly related to the application, that poses requirements of the considered scenes. For instance, Chapter 9 presents a ROS architecture that uses the algorithm for making a robot interacting and explaining scenes to a human supervisor. In such an application we detailed the components used in the architecture that used the semantic interface for processing points clouds and engaging dialogues in a simplified English.

7.6.2 Experimental Setup

Figure 7.1 shows our scenario, where a robot and a user face each other form the opposite side of a table. The robot is an observer that uses a Kinect for processing RGB-D point clout about the table and the objects in it, as shown in Figure 7.5f. On the other hand, the user arranges objects on the table and triggers the algorithm to process new facts for the current instant of time. The objects did not move during the computations of the algorithm, which returned a graph of categories of the scenes at each triggered time. For each node of the output graph, the features matched in the scene and a similarity value were shown to the user, which evaluated if such representation was consistent. Each interval of time needed to provide new facts to the algorithm and obtain an output was considered to be a discrete instant of time.

In order to make the user interacting with the environment without affecting the representation, we gave to him or her the control of the occurrence of discrete instants of time. After the computation performed by the algorithm, he or she was supposed to evaluate the output. Then, after having changed the arrangement of objects on the table, he or she was in charge to trigger the architecture again for evaluating a new scene. We gave to the user the objects shown in Figure 7.5, and we showed to him or her the orientation of a global reference frame ($\hat{x}\hat{y}\hat{z}$), which was the perspective from where scenes where evaluated; as shown in Figure 7.1.

Our algorithm requires facts symbolically represented in the ontology that represent discontinuous quantities, *i.e.* a symbol does not smoothly change from one semantic to another, but its states are discrete. Therefore, small changes of the facts, *e.g.* due to noise affecting perception modules, might be enough for making the algorithm considering drastically different beliefs, which generate inconsistent categorisation of the scene. In other words, with symbols, the algorithm could exploit the semantic of facts, but symbols are not robust to noise. In sections 7.7 and 7.8 we further address this issues that, for our simplified scenario, is solved by given to the user the task to review the perceived information before to trigger the algorithm.

In particular, we mentioned that the user triggered the algorithm, *e.g.* by pressing a button for acquiring a point cloud. After that, the perceiving modules processed the point cloud for segmenting objects, which are shown to the user. The latter was supposed to judge the cluster position and their parameters from a qualitative inspection of the cloud, as presented in Figure 7.5f. If the user found that the data is not consistent with the environment, he or was supposed to press the button for acquiring a new point cloud to be processed. Otherwise, the user was supposed to press another, which triggered the algorithm given facts about the environment that were assumed to be correct.

7.6.3 Object Perception

We deployed in the architecture a perception module that processed a point cloud based on Random Sample Consensus (RANSAC) for identifying the geometrical shape of objects in a table. With the ROS components described in Chapter 2, we perceived objects with a position and a shape, that is used for grounding facts in the robot's representation for the current instant of time. In our scenario, we segmented objects on top of the table, whose are labelled as ether *sphere*, or *plane*, or *cone* or *cylinder*, based on the parameters that RANSAC better fitted in such a clusters of the point cloud.

The OWL reasoner inferred the shape of the objects based on a description of their parameters, which were formalised through a class $\mathbb{R}^3$ aimed in representing 3D points in space (*e.g.* positions and directions) as a vector of three real numbers expressed with respect to the reference frame $\hat{x}\hat{y}\hat{z}$. In accordance with the hierarchy shown in Figure 7.4a, we defined an instance to be an object if it had a centre of mass, which approximates its position,

$$\texttt{OBJECT} \doteq \top \sqcap {=}1\,\texttt{hasCentre}.\mathbb{R}^3. \tag{7.14}$$

A sphere cannot be oriented in space, while the planes, cones, and cylinders has an axis, thus we expressed

$$\texttt{SPHERE} \doteq \texttt{OBJECT} \sqcap {=}1\,\texttt{hasRadius}.\mathbb{R}, \tag{7.15}$$

$$\texttt{ORIENTABLE} \doteq \texttt{OBJECT} \sqcap {=}1\,\texttt{hasAxis}.\mathbb{R}^3. \tag{7.16}$$

A plane is an orientable object defined with a Hessian and four vertexes of three coordinate each, *i.e.* 12 ordered real numbers $\mathbb{R}^{3\times4}$. On

the other hand, a cone or a cylinder has a radius, a height, and an apex or an auxiliary point in the axis

$$\begin{aligned} \texttt{PLANE} \doteq\ & \texttt{ORIENTABLE} \\ & \sqcap =1\, \texttt{hasHessian}.\mathbb{R} \\ & \sqcap =1\, \texttt{hasVertices}.\mathbb{R}^{3\times 4}, \end{aligned} \tag{7.17}$$

$$\begin{aligned} \texttt{CONE} \doteq\ & \texttt{ORIENTABLE} \\ & \sqcap =1\, \texttt{hasRadius}.\mathbb{R} \\ & \sqcap =1\, \texttt{hasHeight}.\mathbb{R} \\ & \sqcap =1\, \texttt{hasApex}.\mathbb{R}^{3}, \end{aligned} \tag{7.18}$$

$$\begin{aligned} \texttt{CYLINDER} \doteq\ & \texttt{ORIENTABLE} \\ & \sqcap =1\, \texttt{hasRadius}.\mathbb{R} \\ & \sqcap =1\, \texttt{hasHeight}.\mathbb{R} \\ & \sqcap =1\, \texttt{hasPoint}.\mathbb{R}^{3}. \end{aligned} \tag{7.19}$$

In the ontology, the reasoner analysed the definition of the above classes and inferred a hierarchy where the leaves are the SPHERE, PLANE, CONE and CYLINDER classes. In accordance with the semantic interface, we considered those classes to be disjointed. This assures a non-ambiguous shape of any perceived objects g_i.

7.6.4 Spatial Relations

In our simplified scenario, we provided the ontology with SWRL rules based on simple geometric computations to infer spatial relationships among pairs of instances (γ_x, γ_y). Based on equations 7.14 and 7.16, we designed crisp spatial relations based on the relative centre of mass of each objects pairs. We decided to keep the following spatial representation simple since our focus is on how to learn new scene categories from human demonstrations, not on scene modelling. Nevertheless, the algorithm supports more sophisticated representations if they can be encoded in the semantic interface.

$\mathbf{R}_1$: *Parallel To.* γ_x and γ_y are parallel, and we write $\texttt{paral}(\gamma_1, \gamma_2)$ *iff.*
- (i) it holds that $\texttt{ORIENTABLE}$: $\gamma_x, \gamma_y, \texttt{hasAxis}(\gamma_x, A_x)$ and $\texttt{hasAxis}(\gamma_y, A_y)$;
- (ii) the vector product between the two normal vectors A_x and A_y is smaller than a threshold $l_1 \in \mathbb{R}^3$.

$\mathbf{R}_2$: *Perpendicular To.* γ_x and γ_y are perpendicular, and we write $\texttt{perp}(\gamma_x, \gamma_y)$, *iff.*
- (i) it holds that $\texttt{ORIENTABLE}$: $\gamma_x, \gamma_y, \texttt{hasAxis}(\gamma_x, A_1)$ and $\texttt{hasAxis}(\gamma_y, A_2)$;
- (ii) the dot product between the two normal vectors A_1 and A_2 is smaller than a threshold $l_2 \in \mathbb{R}$.

$\mathbf{R}_3$: *Coaxial with.* γ_x and γ_y are coaxial, *i.e.* $\texttt{coax}(\gamma_x, \gamma_y)$ *iff.*
- (i) it is consistent that $\texttt{ORIENTABLE}$: $\gamma_x, \gamma_y, \texttt{paral}(\gamma_x, \gamma_y), \texttt{hasAxis}(\gamma_x, A_1), \texttt{hasCentre}(\gamma_x, G_1)$ and $\texttt{hasCentre}(\gamma_y, G_2)$;
- (ii) the distance d between the axes is smaller than a threshold $l_3 \in \mathbb{R}$, *i.e.* $d^2 = (G_1 + A_1 a - G_2)^2 \leqslant l_3^2$, where a can be computed to obtain the minimum distance between the two shapes. It is reasonable to define also a possible coaxial relationship between a $\texttt{Sphere}$ and any $\texttt{ORIENTABLE}$ object when the centre of mass of the first is *close* to the axis of the second. In this case, $\texttt{coax}(\gamma_x, \gamma_y)$ still holds when the $\texttt{paral}(\gamma_x, \gamma_y)$ condition is removed and OBJECT: γ_x holds.

$\mathbf{R}_4$: *Along x.* γ_x and γ_y are aligned along $\hat{x}$, *i.e.* $\texttt{alongX}(\gamma_x, \gamma_y)$ *iff.*
- (i) it holds that $\texttt{OBJECT}$: $\gamma_x, \gamma_y, \texttt{hasCentre}(\gamma_x, G_1)$ and $\texttt{hasCentre}(\gamma_y, G_2)$;
- (ii) the distances between the two centres of mass G_1 and G_2 along the $\hat{y}$ and $\hat{z}$ directions are smaller than thresholds $l_{4\hat{y}}$ and $l_{4\hat{z}} \in \mathbb{R}$, *i.e.* $|G_{1\hat{y}} - G_{2\hat{y}}| \leqslant l_{4\hat{y}} \wedge |G_{1\hat{z}} - G_{2\hat{z}}| \leqslant l_{4\hat{z}}$. Similar relationships are defined for $\texttt{alongY}$ and $\texttt{alongZ}$ (*i.e.* $\mathbf{R}_5$ and $\mathbf{R}_6$, respectively).

$\mathbf{R}_7$: *Right of.* γ_x is at the right-hand side of γ_y, and we write $\texttt{right}(\gamma_y, \gamma_x)$ *iff.*
- (i) it holds that $\texttt{OBJECT}$: $\gamma_x, \gamma_y, \texttt{hasCentre}(\gamma_x, G_1)$ and $\texttt{hasCentre}(\gamma_y, G_2)$;
- (ii) the distance between the two centres of mass G_1 and G_2 along the $\hat{y}$ axis is greater than a threshold $l_7 \in \mathbb{R}$, *i.e.* $G_{1\hat{y}} - G_{2\hat{y}} \geqslant l_7$.

$\mathbf{R}_8$: *Above.* γ_x is above γ_y, *i.e.* $\texttt{above}(\gamma_x, \gamma_y)$ *iff.*
- (i) it holds that $\texttt{OBJECT}$: $\gamma_x, \gamma_y, \texttt{hasCentre}(\gamma_x, G_1)$ and $\texttt{hasCentre}(\gamma_y, G_2)$;
- (ii) the distance between the two centres of mass G_1 and G_2 along the $\hat{z}$ axis is greater than a threshold $l_8 \in \mathbb{R}$, *i.e.* $G_{1,\hat{z}} - G_{2,\hat{z}} \geqslant l_8$.

$\mathbf{R}_9$: *Front.* γ_x is in front of γ_y, *i.e.* $\texttt{front}(\gamma_x, \gamma_y)$ *iff.*
- (i) it holds that $\texttt{OBJECT}$: $\gamma_x, \gamma_y, \texttt{hasCentre}(\gamma_x, G_1)$ and $\texttt{hasCentre}(\gamma_y, G_2)$;
- (ii) the distance between the two centres of mass G_2 and G_1 along the $\hat{x}$ axis is greater than a threshold $l_9 \in \mathbb{R}$, *i.e.* $G_{2,\hat{x}} - G_{1,\hat{x}} \geqslant l_9$.

7.6.5 Semantic Interface and Input Facts

In our implementation, we considered each fact composed by a spatial relation and two objects having a shape. Therefore, we reasoned on scenes that are represented through a semantic interface involving (*i*) characteristics $\mathscr{N}(\mathbf{R}) \sqsubseteq \{\texttt{paral}, \texttt{perp}, \texttt{coax}, \texttt{alongX}, \texttt{alongY}, \texttt{alongZ}, \texttt{right}, \texttt{above}, \texttt{front}\}$, *i.e.* $\mathbf{R}_z \in [1 \dots v{=}9]$, and (*ii*) types $\mathscr{N}(\Gamma) \sqsubset \{\texttt{SPHERE}, \texttt{PLANE}, \texttt{CONE}, \texttt{CYLINDER}\}$, *i.e.* $\Gamma_x, \Gamma_y \in [1 \dots w{=}4]$. We assumed the ontology to contain prior knowledge about spatial relations and shape as presented in sections 7.6.3 and 7.6.4 respectively. We also consistently assumed that Algorithm 7.1 is provided with the related symbols, which are constant names used for retrieving information about facts.

Remarkably, the set of characteristics used in the semantic interface for this scenario did not contains inverse relations (*e.g.* *left* for *right*). With this approach, we reduce the number of facts n than a case where also inverse relations are considered while maintaining the same information about the o beliefs in $\mathbb{E}$, as shown in Section 7.7. However, SWRL rules are computed by the reasoner for all the possible pairs of different objects (xy). Therefore the rules Section 7.6.4 are such too infer symmetric relation when the instances are inverted, *i.e.* for `paral`, `perp`, `coax`, `alongX`, `alongY` and `alongZ`. Also, it is possible to compute the other inverse relations trivially if their semantic is given as prior knowledge in the ontology that is not included in the semantic interface, *e.g.* $\texttt{right}^{-1}(\gamma_x, \gamma_y) \equiv \texttt{left}(\gamma_y, \gamma_x)$. Those might be used for simplifying the interaction with the user by reflecting the scene representation in his or her perspective, *i.e.* through the inverse of all the relations since the user and the robot face each other.

On the other hand, the set of possible types in the semantic interface had to assure a non-ambiguous classification. Therefore the algorithm was provided with a set of names $\mathscr{N}(\Gamma)$ that did not include abstract concepts as `OBJECT` and `ORIENTABLE`. More generally, we considered as possible types T only the leaves of the hierarchy Γ, which are considered to be disjointed.

In our scenario, the architecture was able to identify geometric parameters of objects in accordance with the definition in equations 7.15, 7.17, 7.18 and 7.19. The user reviewed this data and, if it was qualitatively consistent, the user interacted with the architecture for triggering the algorithm. When this occurs, we introduce in the ontology an individual γ_i for each object segmented from the points cloud.

Then, a *preliminary reasoner* task was used for two purposes. On the one hand, each instance was classified in a non-ambiguous type Γ_i. On the other, the spatial relations introduced in Section 7.6.4 as SWRL rules have been evaluated based on the `hasCentre` property that each `OBJECT` is forced to had, as well on the `hasAxis` knowledge, which was

proper of ORIENTABLE instances. Through the preliminary reasoning task, we assured facts in the ontology to be represented in accordance with the semantic interface. Therefore, in our implementation instead of giving input facts to the algorithm, it was enough that the algorithm retrieved the facts F directly from the ontology through the symbols of the semantic interface given as constants.

Nevertheless, SWRL limits the logic and algebraic operations that is possible to perform for asserting facts from perceived data. We want to design an algorithm that does not depends only on such a symbolic language for generating facts. In other words, Algorithm 7.1 do not address the problem of computing facts, which are supposed to be given in a specific format, *i.e.* the semantic interface.

Remarkably, the architecture for our scenario was in charge to perform also another operation in the ontology, which is not specified in Algorithm 7.1, since facts where consider as inputs and not as knowledge already in the ontology. This operation consists in the delete in the ontology each instance γ_i, and consequently each characteristic $\mathbf{R}_z$, when a discrete instant of time ends, *i.e.* when an output graph has been computed.

7.6.6 Representation Complexity

Let u be the number of objects segmented from the table at the current instant of time, *i.e.* individuals in the ontology γ_i, $i \in [1 \dots u]$. In the worst case scenario, each instance characterise the scene with all the relations involving all the other different instances, *i.e.* the possible number of facts could be

$$n \leqslant \left(\frac{v_a}{2} + v_s\right)(u^2 - u), \tag{7.20}$$

where v_a is the number of asymmetric characteristics, which is equal to 3, while v_s is the number of symmetric characteristics, which are 6, *i.e.* $v = v_a + v_s$. In the case of symmetric characteristics, the algorithm needs to represents both the facts $\mathbf{R}_z(\gamma_x, \gamma_y)$ and $\mathbf{R}_z(\gamma_y, \gamma_x)$ since we cannot identify an non-ambiguous type of the instance involved in the range or in the domain, *i.e.* the $\mathscr{T}$ operator is ill-posed, and the perceiving and learning functions at different instant of time could be not consistent. Remarkably, this is not the case for asymmetric characteristics, for which we do not consider the inverse combinations.

Equation 7.20 express the maximum numbers of facts that can be obtained from a general semantic interface. However, not all the facts can be consistent at the same time, *i.e.* two objects cannot be parallel and perpendicular currently. Also, not all the objects pairs are involved in all the characteristics, *e.g.* a sphere cannot be perpendicular to any other shape. Therefore, for a semantic interface required for our spatial

scenario the number of facts n that we perceived at a particular instant of time is typically smaller.

We introduced in Section 7.5.1 that the environmental space $\mathbb{E}$ contains o beliefs spanning in the combinations of reified features $R \times T$, which are applied to each y-th instance occurring in the range of the characteristic of a fact. Therefore, in the worst case the reified relations can describe ϵ with respect to each object. However, similarly to above this is not consistent for some semantic interface, and there could be redundant facts. Thus, in general terms, it is only possible to express that $o \leqslant n$.

Through the mean of the learning function $\mathcal{L}$, we represented categories of ϵ in a space $\mathbb{X}$ that does not depends on the number of objects perceived at the current instant of time u, but only on the size of the semantic interface, *i.e.* a span of $R \times T \times T$. Indeed, Equation 7.8 might generate at most $m \leqslant v \cdot w \cdot w$ category features (144 in our scenario), each of which is defined with a cardinally restriction $k_{zxy} > 0$. Where $m \leqslant o$ if $o \leqslant 144$, since we adopted the open-word assumption.

This implies that the definition of possible characteristics R and types T in the semantic interface fixes the spaces involved in the algorithm. In particular, the algorithm uses its functions for mapping grounded to abstract knowledge by the mean of the semantic interface. In particular, $\mathbb{F}$ contains facts strongly grounded on instances pairs (γ_x, γ_y), while in $\mathbb{E}$ we represent beliefs, which takes into account one instance and one type, *i.e.* Γ_x in the reified characteristic, and γ_y. Finally, $\mathbb{X}$ is a space that does not rely on instances anymore but only on their types Γ_x and Γ_y.

Furthermore, $\mathbb{X}$ contains s categories, where s is the number of times that the learning phase occurred plus one, *i.e.* the root. Namely $s \geqslant 1$, and if $s = 1$ we consider the scene not being recognised.

7.7 Examples and Experimental Assessment

This section discusses some examples of scene based on the tabletop scenario presented in Section 7.6. In particular, it shows the computation performed with the Algorithm 7.1 at some instants of time, which are triggered by the user for presenting to the robot one of the scenes shown in Figure 7.5. The section details the phases of the algorithm to discuss its operations and outputs starting from an empty experience graph. This implies that at t_0 the experience graph in the ontology is represented only by the root $\mathcal{N}(\Phi) = \texttt{EXPERIENCE}$, *i.e.* in Figure 7.4b the classes identifying scenes are not in the robot's knowledge yet.

At time t_1 the user arranged the scene in Figure 7.5a, which was the first experience of the robot that was represented through the input facts F^1 shown in the first cell of the Table 7.1. In particular, this first scene was composed by a `CYLINDER`, a `CONE` and a `PLANE`, *i.e.* $u = 3$ objects, which characterise the facts through spatial relations

time	facts (F)	beliefs (e)	structured category (X_j)
t_1	CYLINDER: γ_1, CONE: γ_2, PLANE: γ_3, paral: (γ_1,γ_2), (γ_2,γ_1), perp: $(\{\gamma_1,\gamma_2\},\gamma_3)$, $(\gamma_3,\ \{\gamma_1,\gamma_2\})$, alongX: (γ_1,γ_3), (γ_3,γ_1), front: $(\gamma_2,\ \{\gamma_1,\gamma_3\})$, right: $(\gamma_1,\ \{\gamma_2,\gamma_3\})$, (γ_2,γ_3).	paralCYLINDER(ϵ,γ_2), paralCONE(ϵ,γ_1), perpPLANE$(\epsilon,\{\gamma_1,\gamma_2\})$, perpCONE$(\epsilon,\gamma_3)$, perpCYLINDER$(\epsilon,\gamma_3)$, alongXCYLINDER$(\epsilon,\gamma_3)$, alongXCONE$(\epsilon,\gamma_1)$, frontCONE$(\epsilon,\{\gamma_1,\gamma_3\})$, rightCYLINDER$(\epsilon,\{\gamma_2,\gamma_3\})$, rightCONE$(\epsilon,\gamma_3)$.	SCENE1 $= \mathcal{N}(\Phi_1) \doteq \Phi \sqcap$ paralCYLINDER • $\geqslant$1 CONE $\sqcap$ paralCONE • $\geqslant$1 CYLINDER $\sqcap$ perpPLANE • $\{\geqslant$1 CYLINDER, $\geqslant$1 CONE$\}$ $\sqcap$ perpCONE • $\geqslant$1 PLANE $\sqcap$ perpCYLINDER • $\geqslant$1 PLANE $\sqcap$ alongXCYLINDER • $\geqslant$1 PLANE, $\sqcap$ alongXPLANE • $\geqslant$1 CYLINDER, $\sqcap$ frontCONE • $\{\geqslant$1 CYLINDER, $\geqslant$1 PLANE$\}$ $\sqcap$ rightCYLINDER • $\{\geqslant$1 CONE, $\geqslant$1 PLANE$\}$ $\sqcap$ rightCONE • $\geqslant$1 PLANE.
t_2	CYLINDER: γ_4, PLANE: γ_5, perp: (γ_4,γ_5), (γ_5,γ_4), alongX: (γ_4,γ_5), (γ_5,γ_4), right: (γ_4,γ_5).	perpCYLINDER(ϵ,γ_5), perpPLANE(ϵ,γ_4), alongXCYLINDER(ϵ,γ_5), alongXPLANE(ϵ,γ_4), rightCYLINDER(ϵ,γ_5).	SCENE2 $= \mathcal{N}(\Phi_2) \doteq \Phi \sqcap$ perpCYLINDER • $\geqslant$1 CONE $\sqcap$ perpCONE • $\geqslant$1 CYLINDER $\sqcap$ alongXCYLINDER • $\geqslant$1 CONE $\sqcap$ alongXCONE • $\geqslant$1 CYLINDER $\sqcap$ rightCYLINDER • $\geqslant$1 PLANE.
t_3	PLANE: γ_{11}, CONE: γ_{10}, CYLINDER: $\gamma_6,\gamma_9,\gamma_{12}$, SPHERE: $\gamma_7,\gamma_8,\gamma_{13}$, paral: $(\gamma_6,\ \{\gamma_9,\gamma_{10},\gamma_{12}\})$, $(\gamma_9,\ \{\gamma_6,\gamma_{10},\gamma_{12}\})$, $(\gamma_{10},\ \{\gamma_6,\gamma_9,\gamma_{12}\})$, $(\gamma_{12},\ \{\gamma_6,\gamma_9,\gamma_{10}\})$, perp: $(\gamma_{11},\ \{\gamma_6,\gamma_9,\gamma_{10},\gamma_{12}\})$, $(\{\gamma_6,\gamma_9,\gamma_{10},\gamma_{12}\},\gamma_{11})$, alongX: $(\gamma_6,\ \{\gamma_9,\gamma_{11},\gamma_{12}\})$, $(\gamma_9,\ \{\gamma_6,\gamma_{11},\gamma_{12}\})$, $(\gamma_{11},\ \{\gamma_6,\gamma_9,\gamma_{12}\})$, $(\gamma_{12},\ \{\gamma_6,\gamma_9,\gamma_{11}\})$, alongY: $(\gamma_{12},\gamma_{13})$, $(\gamma_{13},\gamma_{12})$, front: $(\gamma_{13},\ \{\gamma_6,\gamma_7,\gamma_8,\gamma_9,\gamma_{10},\gamma_{11},\gamma_{12}\})$, $(\gamma_{10},\ \{\gamma_6,\gamma_7,\gamma_8,\gamma_9,\gamma_{11},\gamma_{12}\})$, $(\gamma_8,\ \{\gamma_6,\gamma_7,\gamma_9,\gamma_{11},\gamma_{12}\})$, (γ_6,γ_7),(γ_9,γ_7), (γ_{11},γ_7),(γ_{12},γ_7), above: $(\gamma_7,\ \{\gamma_9,\gamma_{10},\gamma_{11}\})$, $(\gamma_6,\ \{\gamma_9,\gamma_{10},\gamma_{11}\})$, $(\gamma_8,\ \{\gamma_9,\gamma_{10},\gamma_{11}\})$, $(\gamma_{12},\ \{\gamma_9,\gamma_{10},\gamma_{11}\})$, $(\gamma_{13},\ \{\gamma_9,\gamma_{10},\gamma_{11}\})$, right: $(\gamma_6,\ \{\gamma_7,\gamma_8,\gamma_9,\gamma_{10},\gamma_{11},\gamma_{12},\gamma_{13}\})$, $(\gamma_7,\ \{\gamma_8,\gamma_9,\gamma_{10},\gamma_{11},\gamma_{12},\gamma_{13}\})$, $(\gamma_8,\ \{\gamma_9,\gamma_{10},\gamma_{11},\gamma_{12},\gamma_{13}\})$, $(\gamma_9,\ \{\gamma_{10},\gamma_{11},\gamma_{12},\gamma_{13}\})$, $(\gamma_{10},\ \{\gamma_{11},\gamma_{12},\gamma_{13}\})$, $(\gamma_{11},\{\gamma_{12},\gamma_{13}\})$.	paralCYLINDER$(\epsilon,\{\gamma_6,\gamma_9,\gamma_{10},\gamma_{12}\})$, paralCONE$(\epsilon,\{\gamma_6,\gamma_9,\gamma_{12}\})$, perpPLANE$(\epsilon,\{\gamma_6,\gamma_9,\gamma_{10},\gamma_{12}\})$, perpCONE$(\epsilon,\gamma_{11})$, perpCYLINDER$(\epsilon,\gamma_{11})$, alongXCYLINDER$(\epsilon,\{\gamma_9,\gamma_{11},\gamma_{12}\})$, alongXPLANE$(\epsilon,\{\gamma_6,\gamma_9,\gamma_{12}\})$, alongYCYLINDER$(\epsilon,\gamma_{13})$, alongYSPHERE$(\epsilon,\gamma_{12})$, frontSPHERE$(\epsilon,\{\gamma_6,\gamma_7,\gamma_8,\gamma_9,\gamma_{10},\gamma_{11},\gamma_{12}\})$, frontCONE$(\epsilon,\{\gamma_6,\gamma_7,\gamma_8,\gamma_9,\gamma_{11},\gamma_{12}\})$, frontCYLINDER$(\epsilon,\gamma_7)$, frontPLANE$(\epsilon,\gamma_7)$, aboveSPHERE$(\epsilon,\{\gamma_9,\gamma_{10},\gamma_{11}\})$, aboveCYLINDER$(\epsilon,\{\gamma_9,\gamma_{10},\gamma_{11}\})$, rightCYLINDER$(\epsilon,\{\gamma_7,\gamma_8,\gamma_9,\gamma_{10},\gamma_{11},\gamma_{12},\gamma_{13}\})$, rightSPHERE$(\epsilon,\{\gamma_8,\gamma_9,\gamma_{10},\gamma_{11},\gamma_{12},\gamma_{13}\})$, rightCONE$(\epsilon,\{\gamma_{11},\gamma_{12},\gamma_{13}\})$, rightPLANE$(\epsilon,\{\gamma_{12},\gamma_{13}\})$.	SCENE3 $= \mathcal{N}(\Phi_3) \doteq \Phi \sqcap$ paralCYLINDER • $\{\geqslant$3 CYLINDER, $\geqslant$1 CONE$\}$ $\sqcap$ paralCONE • $\geqslant$3 CYLINDER $\sqcap$ perpPLANE • $\{\geqslant$3 CYLINDER, $\geqslant$1 CONE$\}$ $\sqcap$ perpCONE • $\geqslant$1 PLANE $\sqcap$ perpCYLINDER • $\geqslant$1 PLANE $\sqcap$ alongXCYLINDER • $\{\geqslant$2 CYLINDER, $\geqslant$1 PLANE$\}$ $\sqcap$ alongXPLANE • $\geqslant$3 CYLINDER $\sqcap$ alongYCYLINDER • $\geqslant$1 SPHERE $\sqcap$ alongYSPHERE • $\geqslant$1 CYLINDER $\sqcap$ frontSPHERE • $\{\geqslant$3 CYLINDER, $\geqslant$2 SPHERE, $\geqslant$1 CONE, $\geqslant$1 PLANE$\}$ $\sqcap$ frontCONE • $\{\geqslant$3 CYLINDER, $\geqslant$2 SPHERE, $\geqslant$1 PLANE$\}$ $\sqcap$ frontCYLINDER • $\{\geqslant$1 SPHERE$\}$ $\sqcap$ frontPLANE • $\{\geqslant$1 SPHERE$\}$ $\sqcap$ aboveSPHERE • $\{\geqslant$1 CYLINDER, $\geqslant$1 CONE, $\geqslant$1 PLANE$\}$ $\sqcap$ aboveCYLINDER • $\{\geqslant$1 CYLINDER, $\geqslant$1 CONE, $\geqslant$1 PLANE$\}$ $\sqcap$ rightCYLINDER • $\{\geqslant$2 CYLINDER, $\geqslant$3 SPHERE, $\geqslant$1 CONE, $\geqslant$1 PLANE$\}$ $\sqcap$ rightSPHERE • $\{\geqslant$2 CYLINDER, $\geqslant$2 SPHERE, $\geqslant$1 CONE, $\geqslant$1 PLANE$\}$ $\sqcap$ rightCONE • $\{\geqslant$1 CYLINDER, $\geqslant$1 SPHERE, $\geqslant$1 PLANE$\}$ $\sqcap$ rightPLANE • $\{\geqslant$1 CYLINDER, $\geqslant$1 SPHERE$\}$.
t_4	SPHERE: $\gamma_{14},\gamma_{15},\gamma_{16}$, front: $(\gamma_{16},\ \{\gamma_{14},\gamma_{15}\})$, above: $(\gamma_{15},\gamma_{16})$, right: $(\gamma_{14},\ \{\gamma_{15},\gamma_{16}\})$.	frontSPHERE$(\epsilon,\{\gamma_{14},\gamma_{15}\})$, aboveSPHERE$(\epsilon,\ \gamma_{16})$, rightSPHERE$(\epsilon,\{\gamma_{15},\gamma_{16}\})$.	SCENE4 $= \mathcal{N}(\Phi_4) \doteq \Phi \sqcap$ frontSPHERE • $\geqslant$2 SPHERE $\sqcap$ aboveSPHERE • $\geqslant$1 SPHERE $\sqcap$ rightSPHERE • $\geqslant$2 SPHERE.

Table 7.1: The facts, beliefs and learned categories involved in the algorithm when each scene in Figure 7.5 is experienced over consequent instants of time. Beliefs are computed with the perceiving function $\mathcal{P}$ (Equation 7.7), while categories are defined through the learning function $\mathcal{L}$ (Equation 7.8).

always applied among objects pairs (γ_x, γ_y). In Table 7.1 we simplify the DL notation using squared brackets for grouping combination of objects pairs. For instance, the fist cell expresses that $\texttt{front}(\gamma_2, \gamma_1)$ and $\texttt{front}(\gamma_2, \gamma_3)$ was feasible facts for the scene at time t_1. Therefore, we obtained n=13 input facts, whose describe γ_y as a characteristic of γ_x, *e.g.* the last fact of the first cell in Table 7.1 represents that "γ_2 *has* γ_3 on the right-hand side".

The perceiving phase of Algorithm 7.1 started when the ontology contained facts that respected the semantic interface (Equation 7.6). This occurs at time t_1, after that (i) the geometric parameters of objects have been perceived by the robot and checked by the user, (ii) the related instances were introduced in the ontology, and (iii) the preliminary reasoning task (presented in Section 7.6.5) was performed to generate spatial relationships.

In this phase, we used the perceiving function $\mathcal{P}$ to create beliefs demonstrated at time t_1, which are shown in the second cell of the first row of the table, which contains e^1. Similarly as before, in the table we simplify the formalism, *e.g.* the two beliefs $\texttt{frontCONE}(\epsilon, \gamma_1)$ and $\texttt{frontCONE}(\epsilon, \gamma_3)$ are expressed in a single line, which is a fragment of the representation of the current scene that can be read as "in ϵ a CONE has γ_1 and γ_3 in front of it". For the scene in Figure 7.5a, we appled Equation 7.7 to generate o=13 beliefs from all the perceived facts by the mean of the perceiving function $\mathcal{P}$. For this scene, the number of facts and beliefs was equal, *i.e.* n=b, which represents that the inputs do not contain any redundant facts. Remarkably, the number of facts generated with the semantic interfaces introduced in Section 7.6 was considerably lower than the worst case, where n would be 45 (Equation 7.20).

Algorithm 7.1 continued with the classification phase at time t_1, which did not generate any outputs since the experience graph contained only the root, *i.e.* s=1. Therefore, the learning phase occured, and $\mathcal{L}$ (Equation 7.8) was used to generate a new scene category $\Phi_\epsilon \in \mathbb{X}$. Then, the OWL reasoner performed the structuring function $\mathcal{S}$, which arranged the learned category Φ_ϵ as a new node of the experience graph, *i.e.* $\Phi \sqsupset \texttt{SCENE1}$=$\mathcal{N}(\Phi_1)$, which is defined in the first row and third column of Table 7.1. For instance, in that cell of the table, the category is defined also with the $\geqslant$1 `rightCYLINDER.CONE`, $\geqslant$1 `rightCYLINDER.PLANE` restrictions, which represents two features that can be read as "a scene is of the Φ_1 category if [...], and at least 1 CYLINDER has a CONE on the right-hand side, and at least 1 CYLINDER has a PLANE on the right-hand side, and [...]".

After the learning phase, the algorithm performed the classification phase again for instant t_1. The experience graph contained only the root and the Φ_1 category (*i.e.* s=2,), which was retrieved through the query in Equation 7.9 since all the features of the category were matched in the beliefs about the scene. Therefore we obtained an

output graph containing the root and the learned category, which had a cardinality ${}^1d{=}13$, *i.e.* the sum of the restrictions k_{zxy} for all the features of Φ_1 in accordance with Equation 7.11. We observed that the similarity value was equal to ${}^1d_\epsilon{=}13/o{=}1$ (Equation 7.12), since all the beliefs of ϵ matched all the features of Φ_1 because the latter had been generated from the former through $\mathcal{L}$. Then, the algorithm returned the output graph for the instant t_1, which was assessed by the user.

At time t_2, the user interacted with the buttons for acquiring consistent facts for the scene in Figure 7.5b. Table 7.1 shown the $n{=}5$ input facts of F^2, which involved a `CYLINDER` and a `PLANE`. Those facts were used to compute the $o{=}5$ beliefs of e^2 through the perceiving function $\mathcal{P}$. Then, the classification phase of the algorithm started, and the reasoner of the ontology was queried in accordance with Equation 7.13. The response of the query did not contain any categories apart from the root of the experience graph, *i.e.* $\Phi^* \equiv \Phi$, and $s{=}1$. This occurs because all the features of the only category in the experience graph (Φ_1) were not matched with the beliefs about the scene ϵ at time t_2. Since $s{=}1$, we considered the scene not classified, and the learning phase occurred again.

During the learning phase at time t_2, the category `SCENE2`$=\mathcal{N}(\Phi_2)$ was generated and structured in the ontology through $\mathcal{L}$ and $\mathcal{S}$. In accordance with the cell in the last column and second row of Table 7.1, Φ_2 is defined with $m{=}5$ features, each having one cardinality restriction. Based on those, the OWL reasoner structured the new category to be a parent of Φ_1, since all the features of the new category were also respected from the one learned at t_1, *i.e.* $\Phi_1 \sqsubset \Phi_2 \sqsubset \Phi$

The classification phase was performed again at time t_2, and we query the reasoner (Equation 7.9). The query is solved with a hierarchy containing the root Φ, and the category learned at the current instant of time Φ_2; thus $s{=}2$. Similarly to the previous instant of time, we observe that Φ_2 is recognised with a similarity value ${}^2d_\epsilon{=}1$, *i.e.* it perfectly categorised the scene. This is always the case when the algorithm terminates after the learning phase since we match the feature generated form $\mathcal{L}$ against the same beliefs that have been used from the learning function, *i.e.* introduced in the ontology from $\mathcal{P}$ at the same instant of time.

At the time instance t_3, the robot processed the scene in Figure 7.5c, and the algorithm generated the knowledge in the third row of Table 7.1. This scene involves $u{=}8$ objects that characterise the scene through $n{=}98$ facts contained in F^3. For the current instant of time, the perceiving function $\mathcal{P}$ is used to compute $o{=}60$ beliefs in B^3; therefore, $\mathcal{P}$ generates 38 redundant beliefs based on the semantic interface we adopted. For instance, redundant knowledge were generated when the combinations $\texttt{paral}(\gamma_6, \{\gamma_9, \gamma_{10}, \gamma_{12}\})$ and $\texttt{paral}(\gamma_9, \{\gamma_{10}, \gamma_{12}\})$ are considered based on `CYLINDER`: γ_6, γ_9.

From those facts, $\mathcal{P}$ computes the $\texttt{paralCYLINDER}(\epsilon, \{\gamma_9, \gamma_{10}, \gamma_{12}\})$ and $\texttt{paralCYLINDER}(\epsilon, \{\gamma_6, \gamma_{10}, \gamma_{12}\})$ beliefs respectively. In this case, the combinations (ϵ, γ_{10}) and (ϵ, γ_{12}) associated to the reified characteristics paralCYLINDER are duplicated and considered only once, since duplicate them in the ontology would not further affect OWL reasoning.

During the classification phase at time t_3, the query to the reasoner (Equation 7.9) was solved with the hierarchy containing the root as well as the Φ_1 and Φ_2 categories. Nevertheless, those categories were too simplistic models of the scene, and we expected the similarity value to be low. Indeed, ${}^1d_\epsilon{=}13/o{=}.21$ and ${}^2d_\epsilon{=}5/o{=}.08$, which was lower than the threshold $\ell{=}.4$. in accordance with $\mathcal{C}$ (Equation 7.13), the two classes were not included in the output graph Φ^*. Therefore, the scene was considered not recognised at time t_3, and the learning phase occurred. Table 7.1 shows the results of the learning function that generated a new category $\texttt{SCENE3}{=}\mathscr{N}(\Phi_3)$, which was structured through $\mathcal{S}$ in a new experience graph where $\Phi \sqsupset \Phi_2 \sqsupset \Phi_1 \sqsupset \Phi_3$. Then, the classification phase occurred again, and the algorithm proposed to the user an output graph containing only Φ_3, which was classified with a similarity value ${}^3d_\epsilon{=}1$, *i.e.* the sum of the cardinality restrictions of each $m{=}40$ features was ${}^3d{=}60$. As above, the other two categories Φ_2 and Φ_1 were not nodes of the output graph since their similarity values were lower than the threshold, and the related edges are removed accordantly from the experience graph. Therefore the output graph contains only two nodes, *i.e.* $\Phi \sqsupset \Phi_3$.

The scene in Figure 7.5d was evaluated at time t_4. Similarly to the previous instants, Table 7.1 shows by column the $n{=}5$ facts in F^4, the $o{=}5$ beliefs e^4, and the category $\texttt{SCENE4}{=}\mathscr{N}(\Phi^4)$ with cardinally ${}^4d{=}5$. As occurred at time t_2, the reasoner could not classify ϵ in any categories of the experience graph. Therefore, the learning phase occurred and the algorithm returned only one category, which perfectly described the scene, *i.e.* ${}^4d_\epsilon{=}1$. At the current instant of time, we observe that during the structuring function the reasoner inferred that the scene had some similarities to the scene experienced at time t_3. Indeed, as occurred at time t_2, the new category Φ_4 was a parent of Φ_3 since all the restrictions of the latter were also respected from the former. This happened because in both scenes (figures 7.5c and 7.5d) there were three spheres placed on the table from left to right, and from front to behind.

At time t_5 the user arranged the scene in Figure 7.5e. Since we adopted a semantic interface concerning crisp spatial relations, the algorithm could not reason on how far on the right-hand side an object is with respect to another instance. Thus, it was not possible to discriminate the differences with respect to the scene experienced at the previous instant of time (Figures 7.5d). Indeed, the scenes at time t_4 and t_5 were represented with the same input facts, which

consequently generate equivalent beliefs and classification with the highest similarity value, in accordance with Table 7.1. Therefore, the output graph at time t_5 contained the Φ_4 category, classified with a similarity value ${}^4d_\epsilon{=}1$ without performing the learning phase.

At time t_6 the algorithm was triggered with an experience graph $\mathbb{X}^{t-1}$ as shown in Figure 7.4-(b). In accordance with Section 7.5.3, we observe that a deeper node in the experience graph is related to the category of a more complex scene, *i.e.* the number of restrictions ${}^jd$ increases. Also, we observed that categories were structured independently from the learning time and that a category could be represented as containing the union of others, *e.g.* $\Phi_3 \sqsubset \Phi_4$ and $\Phi_3 \sqsubset \Phi_1$.

The scene arrange by the user at the time t_6 was Figure 7.5a again. After the perceiving phase, the ontology contained the same beliefs represented at time t_1, but now the classification function $\mathcal{C}$ returns an output graph Φ^* without triggering the learning phase since the robot did already experience such scene. The output contained Φ_1, which was classified with the highest similarity value since the beliefs F^1 and F^6 were equivalent. Nevertheless, the output graph did not contain Φ_2, even if it is a category that consistently solves the query made during the classification phase because ${}^2d_\epsilon{=}.38 < \ell$, since only 5 features over 13 beliefs matched.

In contrary, if a time t_7 a scene equivalent to Figure 7.5c without the cone γ_{10}, was evaluated. Then, a new category $\Phi_7 \sqsupset \Phi_3$ will be generated in the ontology with $o{=}40$ beliefs from the learning phase. If at a time t_8 a scene equivalent to Figure 7.5c is presented again, the algorithm will return an output graph containing Φ_3 with the highest similarity value by definition, and Φ_7 with a similarity value $d^7_\epsilon{=}40/60{=}.66 > \ell$.

7.8 Discussions

In the previous section, we show the computation that the algorithm performed for learning a structure scene categories, each with a single demonstration. In particular, Table 7.1 shows the models of scene categories that the robot built at runtime, which were evaluated by an OWL reasoner. Through our examples, we observed that the structure that it was possible to learn is limited to inclusions among scenes ($\sqsubset$). In particular, the presented algorithm cannot represent scene overlapping, since all the features of a category must also be satisfied by another category for being related in the experience graph. In other words, represent a relation among two categories that share only a part of their features in the ontology is still an open issue for our algorithm. Nevertheless, if the scene that contains only the features in common with those two categories is shown to the robot, it would create a node in the experience graph that relates such two categories,

e.g. as occurred for Φ_3 in our examples, which could be seen as the overlapping of the Φ_1 and Φ_4 categories.

In the proposed implementation, the algorithm generates a not leaner number of facts with respect to the number of objects. Indeed, the number of facts n depends on the number of objects u quadratically, as shown in Section 7.6.6. Since we classified categories based on a threshold of the similarity value based on n, a pair of scenes differing from one single object will be more similar between each other if they contain many other objects than the case in which few objects are on the table.

For this reason, in Section 7.7 the scenes in figures 7.5a and 7.5b (that differ only for the cone) were considered to be substantially different at time t_7. For the same reason, at time t_8 we observed that Figure 7.5a would be similar to the same scene but without the cone. This is consistent in our scenario since we wanted a semantic interface that allows the robot to reason only on spatial relationships among objects. For different scenarios and consequently different semantic interfaces, it is possible to linearise the computation of the similarity values in specific ranges of u. On the other hand, it is also possible to change the thresholding operation with more sophisticated functions that, for instance, depends on the number of objects on the table at learning time.

Remarkably, the definition of the semantic interface for a real scenario could be not trivial since it should assure the correct behaviour of the algorithm. For instance, having characteristics that are exclusive (*e.g.* `paral` and `perp`) reduce the overall number of facts required for representing a scene and the ambiguities as well as the computation complexity. Also, having characteristics that are asymmetric reduces the number of facts, and consequently the complexity. Furthermore, semantic interfaces that rely on types that uniquely define an object (*e.g.* the `CONE` or the `PLANE` in Figure 7.5) prevents redundant beliefs and improves the output of the algorithm. This occurs because the algorithm does not track the relationships between objects in the same scene. Indeed, while facts pairs instances γ_i, categories are based on their type Γ_i, and if more instances are of the same type, we lose the direct representation of their relationships. In other words, a learned category might express that a generic cylinder should have on the right-hand side a generic sphere. Not that a particular cylinder should have a particular sphere on the right-hand side. Therefore, the category cannot take into account further characteristics that such particular objects had at learning time.

We based our algorithm on a purely symbolic representation since we wanted to parameterise it for a reach set of scene types, which are represented through symbolic facts spanning in a space fixed by the semantic interface. Such a representation of facts is automatically evaluated by an OWL reasoner, which also supports logic representation

for considering more complex definitions of the types and characteristics used for representing each fact. For instance, it could be possible to encode also the colour or dimensions in the definition of the types in Γ, *e.g.* `YELLOWSPHERE` or `BIGSPHERE`. On the other hand, it is possible to rely on perception modules that can assign an exhaustive tag to objects, *e.g.* `TENNISBALL`.

On the one hand, symbols in the ontology help for the interface with many components in a robotic architecture and are communicable to a human. On the other, they make the algorithm not robust to perturbation, *e.g.* perception noise. This because in the ontology symbols do not smoothly change from a state to another; therefore even if only one fact is not consistent with the environment, the scene that the algorithm evaluates will be drastically different from the one in the environment. For instance, if the object γ_{10} in Figure 7.5c is misperceived as a cylinder, then the algorithm would learn a category that is unrelated from all the others in the experience graph, *i.e.* in Figure 7.4 there would be a new category structured as a direct child of the root. On the contrary, if γ_{10} is correctly perceived, the algorithm would classify the scene as Φ_3, which contains Φ_2 and Φ_1.

For solving the issue due to perception noise, we made the user a supervisor that filters perceived facts that are not consistent with the environment. However, this could not be feasible for some applications, but with the current crisp implementation of the algorithm, handle uncertainties is an open issue. Furthermore, we presented an algorithm where the learned scenes are enumerated based on time, but for real application, they should have a meaningful name that is retrieved during classification. In this case, we argue for an architecture that involves the user during the assignment of the scene name $\mathcal{N}(\Phi_\epsilon)$ during demonstrations.

We observed that a user could understand our symbolic representation, but for a complex scene this is not an easy task. To simplify the interaction, we argue for a semantic interface that spans a limited number of facts, or for methods that select the most relevant features to be communicated to the user. Unfortunately, the representation of spatial relations in a human manner is not trivial, and our simple implementation presented in Section 7.6.4 is far from being satisfactory during the interaction, *e.g.* we observed that our implementation of `front` is misleading. Nevertheless, our semantic interface allows for straightforward integration of perception modules for generating different sets of symbolic input facts, also in a multimodal manner as presented in Chapter 4.

In our scenario, we used a semantic interface that represents the qualitative characteristics of scenes, *e.g.* `right`. Consequently, the categories that the algorithm could learn qualitatively classify the scenes, which is Section 7.7 where the scenes in figures 7.5d and 7.5e are considered to be qualitatively equivalent. Nevertheless, it is possible

to use also semantic interfaces spanning facts with more quantitative characteristics, *e.g.* `hasColour`(γ_x, γ_y), where `CONE`: γ_x and `RGB`: γ_y. This is coherent with our semantic interface since we do not limit the types and characteristics to be of the same domain, but they could span facts between instances with different semantics (*e.g.* `OBJECT` and `COLOR`). Therefore, it is possible to defines also facts between objects and numbers, *e.g.* a characteristics `hasVolume`(γ_x, γ_y), where γ_y is a real number. Nevertheless, through characteristics that relatively describes facts (*e.g.* `hasVolume`(γ_x, γ_y), where `BIG`: γ_y) it is possible to exploit symbolic reasoning more efficiently. However, such type of reasoning implies qualitative knowledge, depending on the level of details used in the symbolic representation, *e.g.* the range of number represented as a `BIG` volume.

As far as computational aspects are concerned, the whole algorithm requires an average time of 3.272 seconds on an Intel i7@3.4 GHz with 8 Gb RAM. Although with the presented examples we could not characterise the computational performances of our prototyping implementation exhaustively. We observed that the algorithm could be used within soft real time constraints suitable a for human-robot interaction scenario. Indeed, we argue room for computational improvements of our prototyping implementation, especially in if an *ad hoc* reasoner is used for maintaining knowledge in the described structures based only on a particular semantic interface. Nevertheless, the design of the interface for a real application is far from trivial, and we need to test different semantics to be compared via experiments. With investigating purposes, we designed the algorithm to support an expressive and flexible semantic interface.

7.9 Conclusions

The paper details an algorithm that performs structured learning of categories of scenes based on human demonstration; we called it the Scene Identification and Tagging algorithm (SIT)[7.3]. The algorithm is based on an OWL ontology where we represent knowledge in three different spaces based on a semantic interface. The semantic interface is composed of two sets of symbols that represent the possible facts that can define a scene at runtime. Based on the facts, the algorithm learns and structures scene categories, *i.e.* an experience graph, which is used for computing an output during the classification phase. The output is a graph contains only categories that describe the environment, and they are ranked with a similarity value.

7.3 https://github.com/EmaroLab/scene_identification_tagging.

We present a general-purpose algorithm that we deploy in a ROS-based architecture, implemented for a simplified scenario aimed to highlight the behaviour of the algorithm. In the referenced scenario the architecture segments a point cloud and compute crisp spatial relations among objects with a geometrical shape. Also, the architec-

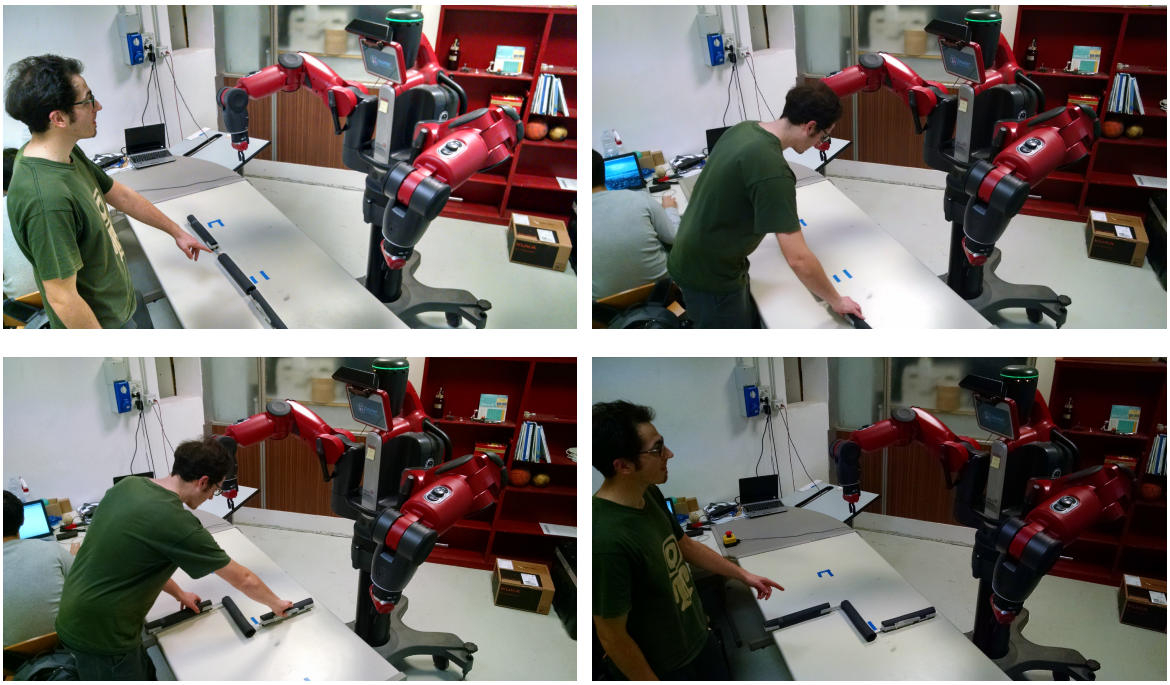


Figure 7.6: Showing configurations of an articulated object.

ture relies on a user that arranges the scene, supervises the perception modules, and triggers the algorithm. Nevertheless, we discussed our design based on the semantic interface that makes the algorithm very flexible by the mean of symbolic representations. This type of interface assumes on the one hand that the ontology contains facts as symbols having a semantic representation, which is given as prior knowledge. On the other, that the algorithm has to be aware of some of those symbols. In Chapter 10 we exploited different semantic interfaces based on the referenced spatial scenario to implement an architecture that maintains an autobiographical memory for a robot, provided with consolidating and forgetting abilities. That paper is based on the fact that our algorithm do not fixes the representation to be used. It only requires semantic instances of a non-ambiguous type, paired through relations, *e.g.* `bigger(T1, C1)` with `TRACK: T1` and `CAR: C1`, or `before(Breakfast,Toothbrushing)` with `ACTIVITY: Breakfast, Toothbrushing`.

Given symbols the semantic interface, the algorithm performs one-shot learning of scenes category, which is based on cardinally restrictions and reification of axioms in the ontology. We exploit this feature for learning the category classifying the current scene if the latter has not been experienced previously. Therefore, our algorithm always provides a classification of a presented scene. The learned categories are structured over time, and the classification might contain more categories, which are returned as an output graph. In particular, the output graph contains all the categories of included scenes (*i.e.* sub-

scenes) previously experienced that match the current scene. This makes the learned representation suitable for representing repetitive patterns, *e.g.* to represent a topological map where many equal offices are connected to the same corridors in each floor of a building.

At the same time, the representation that our algorithm can learn is not suitable for overlapping scenes. Nevertheless, we discussed how the interaction with a user might guide a robot to learn a suitable set of categories for an application, which is automatically aggregated by the algorithm. For doing this, we exploit the symbols of our representation again since they are understandable by humans. In particular, we use a simplified human-robot interface to explain the user the experience graph, which was maintained by the algorithm for some demonstrations.

The drawback of having a purely symbolic and crisp representation of knowledge is that it is not robust to uncertainties. Therefore, small perturbation, *e.g.* perception noise, substantially affect the soundness of the learning structure. This occurs because symbols do not smoothly change from one state to another, and a small perturbation on row data might result on considerably changes in the related symbols, *i.e.* a `CONE` is miss-classified as a `CYLINDER`. For solving this issue in our use case, we rely again on the user that can understand if a symbol is not consistent with the environment. This allows as to assure correct inputs to the algorithm, but we argue its limitation in a real scenario.

As further work we want to focus on three aspects. We want to investigate suitable methods for an effective human-robot interaction considering, for instance, the learning of structured planning tasks; an example is shown in Figure 7.6. Also, we want to evaluate different semantic interfaces in human activity recognition scenarios based on [83], as well as for a more sophisticated spatial representation. Finally, based on fuzzy ontology [24], we present in Chapter 8 an extension of the algorithm able to deal with uncertainties

8

One-Shot Structured Learning of Robot's Experience in Uncertain Environments

Abstract

This paper presents an extension of the Scene Identification and Tagging algorithm (SIT) for representing uncertain knowledge. SIT performs one-shot learning of categories that are structured in an experience graph for reasoning and classifying purposes. The algorithm is based on an ontology and exploits logic reasoning for structuring experiences based on symbolic inputs. It has been designed for a Human-Robot Interaction scenario and allows the robot to directly share its beliefs with users based on qualitative relations. We preliminary tested our algorithm in a tabletop scenario concerning objects spatially arranged, and we evaluate the resulting structured knowledge compared with our previous implementation. This paper discusses and motivates the use of the algorithm also for more complex applications.

structured learning · robot memory · uncertain environment · fuzzy ontology

Authors

Luca Buoncompagni,[♠] Fulvio Mastrogiovanni[♠] and Alessandro Saffiotti[♣]

Affiliation

[♠]University of Genoa, [♣]Örebro University.

Paper to be peer-reviewed.

8.1 Introduction

Nowadays robot can rely on sophisticated models for recognising salient features of the environment, as well as for reasoning in complex structures. Those functionalities are crucial to act in an unsupervised environment, and usually requires difficult development steps concerning both supervised training and knowledge engineering. Some systems exploit reinforcement learning for making the knowledge of the robot evolving but, due to a large number of required trial, as well as the incommunicability of the models they generate, this approach could be not suitable for a high-level Human-Robot Interaction (HRI) scenarios. In particular, if a robot and a user wants to exchange abstract and qualitative beliefs about the observation of the environment, they would require to develop a deep understanding between each other.

For the case of natural language processing, [169] presents a knowledge bootstrapping for representing, in the robot's knowledge, the semantics and the similarities to be inferred for planning and acting purposes based on affordance. In this paper, we foresee robots involved in knowledge bootstrapping phase, in which a user demonstrates to the robot the environment in some particular states. The robot should (*i*) learn a representation of such a state, (*ii*) be able to classify it in the future taking uncertainties into account, (*iii*) reason about the implications of observation concerning previous experience, and (*iv*) be able to communicate its beliefs to a user.

In a real scenario, the salient features of the environment are characteristic or events occurring in the robot surrounding. Those typically involve a small part of the environment, *e.g.* few object or actions, that the robot is supposed to detect because they provide useful information to be processed. For instance, if a human and a robot interact during an assembly task, the presence of the user in a dangerous position might trigger some robot behaviour for safety reasons. On the other hand, there would be safe situations in which the robot should take other decisions; for instance, approaching the user. In such an application, the robot should either be able to contextualise the information it has been processed, or it should be able to perceive not only objects but also to their relations and all the other useful information for representing the context, at each time instant. In the former case, the robot would need structures to reason on knowledge it collected. On the latter instead, it should be provided with a perception module able to assess an expressive representation, *i.e.* yet another type of structure instead of simple labels.

We foresee robots that, after some bootstrapping demonstrations, maintain a structure of their experiences with the purpose to recognise salient patterns in an uncertain environment, *i.e.* qualitative sub-scenes. In HRI application, we argue for robots that can explain their beliefs

to a user with the purpose of reducing the interaction gap [81]. Also, we aim to design a system that automatically stores and reasons on different situations and their implication through demonstrations. Although this paper presents an algorithm tested in a preliminary scenario, our future objective is to deploy it in more complex applications, which involve large or distributed knowledge representations and the computational complexity they demand, as well noises and uncertainties.

In this paper, we extend the the Scene Identification and Tagging (SIT) algorithm presented in Chapter 7, which maintains symbolic and structured knowledge over time. The latter processes scenes and implements a one-shot learning strategy based on a logic formalism to encode experience in an ontology. In this formalism, the algorithm exploits standard reasoning techniques to classify scenes in previous experiences, as well as for reasoning on their implication. At runtime, the algorithm generates and maintains a graph of models describing experienced scenes, through which the current state of the environment can be assessed. In particular, the graph maintained by the algorithm allows to classify a small part of the experienced scene, *i.e.* it generalises models stored through observations. In other words, the algorithm represents sub-scenes based on a representation of their complexity.

The algorithm also provides with a general interface for maintaining experiences graphs based on the specific peculiarities of the environment that different application might involve. Since the algorithm performs purely symbolic reasoning, we could implement such an interface at the semantic level, and in Chapter 7 we discussed the benefits and drawbacks of this design. Therefore, the algorithm for the perception modules, used to generate the inputs, can be implemented independently from the chosen semantic; however it will require the knowledge representation as defined in the syntactical interface. On the other hand, symbolic knowledge is not directly measurable, and it might require sophisticated perception techniques. Also, the crisp symbols we used in Chapter 7 were not robust to perturbations since the symbols were not smoothly changing from one state to another. Nevertheless, with a certain representation of the inputs to the algorithm, we showed that it could reason on qualitative implications, and it also supports sub-symbolic formalisms and quantitative reasoning.

In a typical HRI scenario, there is the evidence of at least two sources of uncertainties. Firstly, perception is subjected to noise, and therefore the beliefs that a robot has about the environment are disturbed but – we suppose – not substantially corrupted. The other source of uncertainty comes from users since their behaviour is unknown, as well as their beliefs. In particular, if the user should demonstrate scene and supervise the robot while it makes experiences, it is essential to establish a duplex communication between he or she and the robot.

For instance, if the user wants to teach a scene to the robot based on the information that a "keyboard is connected to a mouse", it is crucial that both of them agrees on the semantics of such information. Different from perceived information, the uncertainties about this knowledge could be higher since different users might describe the same environment in many different ways, as we presented in Section 1.1.

We argue that a robot with a knowledge structured in a fashion that is familiar to the user might considerably simplify the communication issues if the robot should explain its beliefs, or if the user wants to correct them, *i.e.* if they want to share knowledge actively. In Chapter 9, we discus an application of the SIT algorithm deployed in a scenario where a user supervised the robot beliefs through dialogues. In that work, we exploited the Description Logic (DL) [11] formalism for sharing experiences with a semantic that was familiar to the user while respecting the syntactic requirements to interface a dialogue module with the algorithm in the Robotic Operative System (ROS). Especially for dialogues, we do not want to assume the user to prove the robot with quantitative knowledge but, as persons would naturally do, he or she should use qualitative information. Knowledge extracted from qualitative information is vague *per se*, especially if used in different contexts; therefore, the robot has to be able to reason with uncertain knowledge.

In this paper, we present an extension of the SIT algorithm such to represent knowledge in a fuzzy ontology where the robot could reason with noisy data and uncertain representations. In particular, the paper details the implementation of the algorithm and draws a comparison between our previous crisp implantation. To enforce the comparison we tested the fuzzy implementation of SIT in the same scenario presented in Chapter 7, where objects were spatially arranged on a table. In this preliminary scenario, we observed that the fuzzy implementation fully extends the crisp algorithm, which maintains all its features and provides more sophisticated experiences graphs and, consequently, more expressive classification outcomes.

In the next section, we introduce the background, related work, and contributions. In Section 8.3, we reformulate the problem presented in Section 7.4 for a fuzzy knowledge representation, and in Section 8.4 we discuss all the operations performed by the algorithm. Section 8.5 provides some details about the implementation that have been used to collect some preliminary results presented in Section 8.7, which are discussed in Section 8.6.

8.2 Related Work

Also in the robotic domain, outstanding results have been obtained through data-driven techniques since they perform well for many tasks

that a robot should accomplish. For instance, convolutional neural networks have been largely used for objects recognition, *e.g.* from points clouds [37], detect on-line gestures [33], as well as posture [100] and activities [128], just to name a few. Data-driven techniques estimate the probability distribution for generating models that are robust to noise and well fuse row data of different nature. This property resulted in being suitable for motion planning; for instance, in [152] Deep Reinforcement learning has been used for implementing a map-less motion planner for mobile robots, while in [95] deep learning is used to control a robotic arm during grasping tasks. The knowledge that can be perceived with a trained model is typically instantaneous and simple, *e.g.* a label, or a vector of numbers. Nevertheless, other works show that a combination of those outcomes with semantic representation can produce more expressive and accurate results. For instance, [96] shows a combination of spatial relations for structuring labels that their models generated from images, *e.g.* classifying statements as "boy holding stick".

Apart from behaviour-based robots, it seems evident from the literature that complex actions require some planning or reasoning, which should be supported by expressive symbolical, or sub-symbolical, structures. STRIPS-like planner [60] are based on symbols with specific semantics, which are used to reason on the sequence of actions for obtaining a final state of the environment from an initial configuration. Other sub-symbolic planners, as the Hierarchical Task Network, compute the actions to be performed to accomplish a task based on its decomposition in sub-tasks [90]. Also, probabilistic representation has been used with planning purposes, based for instance on decision Markov processes [106], which can be robust to uncertainties, and are suitable for parameter learning. Nevertheless, also works aimed at learning symbolic plans have been proposed. For instance, [118] uses an experience graph, which contains plans performed through demonstration and auxiliary information, to improve the performances during planning. While, in [110], inferences about the goal are used for learning and generalising plans. Remarkably, an open issue that the authors highlighted concerns the possibility to reason about task similarities.

A typical approach in robotics is to use data-driven techniques as an interface between raw data and higher sub-symbolic layers supporting the reasoning. For instance, Knowrob [156] has a hybrid reasoning core which uses heterogeneous data types and algorithms, which are interfaced to an external layer through logic languages. Knowrob can be used in a ROS architecture through perception-action loops, and it supports learning as well as internal simulations. Similarly, ROBOBRAIN [134] is a framework based on conditional random fields used to learn nodes in a graph that represents the environment semantically, *e.g.* objects, colour, affordance and activities. Thanks to

a sub-symbolic representation, it effectively deals with continuous data such as trajectories [79]; this knowledge representation can also evolve based on new demonstrations through non parametric learning. Observations are used to reason among different domains, which provide general knowledge through queries in a simplified natural language.

During the interactions, the robot should rely on raw sensor data to assess the environment as well as the user. At the same time, the user should be able to understand the decision and intentions of the robot. If the the robot and the user do not understand each other, the deployment of end-to-end learning techniques for building models becomes limited. The problem of making the robot able to explain its knowledge for full interacting with the users is still a state of the art challenge [140].

Also, learned models are typically neither modular nor portable since any changes of the robot behaviours and capabilities would require a complete retraining and validation process. In contrast, symbolic formalism can be used to generate modular structures that are familiar to the user; but these structures are affected by a trade-off between performance, especially as far as scalability is concerned, and expressibility [29]. However, since data-driven approaches are based on probabilistic estimation, the model they generate will deal with uncertainties, while symbolic formalism based on deterministic logics are typically not robust to noise.

We aim at building a system that can structure its knowledge based on demonstrations, which implies perception and elaboration of raw data for the purpose of learning instead of behaving. After an initial training phase where the robot should have gained experience, its knowledge should be recalled for acting based on further inputs. We propose a system that learns and represents data symbolically in order to simplify the interaction of the user when supervising – not physically – the robot while learning. The outcome of the system should be a set of structured symbols with an associated semantic describing also uncertainties. Since we want to build representations that can interface with the highest number of different software components as possible, we decided to represent the robot's knowledge using the Ontology Web Language (OWL) standard [108].

OWL represents knowledge through DL formalisms in an ontology where reasoners can perform instance and consistency checking with inferring purposes, *e.g.* Pellet [144]. Such a representation is deeply structured, and expressive queering languages can be used to retrieve the knowledge. For this reason, ontology has mainly been used in robotic with the purposes to store sensory data based on some prior knowledge consistently. For instance, OWL is used in [105] for multimodal interaction in HRI industrial scenarios, while in [32] consistency checking has been used for planning adaptation, and in [166] OWL

is used for sharing semantics of objects in a map with humans. Also, ontology has been used for a simple and direct verbal communication with users but, as highlighted in [31], it could be not suitable for a straightforward interpretation of temporal depended knowledge, as well as knowledge with a subjective degree. Typically, those applications use statically defined ontology, and the reasoning abilities of the robot are limited to such prior knowledge. Moreover, the design of such an ontology usually involves a complex development and evaluation. In particular, it is crucial to identify the semantic symbols that can well describe the environment through qualitative properties, *i.e*. design the knowledge with an accurate level of detail. If it is too specific, the complexity will increase considerably without adding any substantial value to the knowledge that the robot needs. On the contrary, if it is too abstract, ambiguous representation would occur.

Algorithms able to identify symbolic definitions of structured knowledge through observations have been proposed. Usually those algorithms address the problem of mining a large amount of data and do not consider issues related to robotic applications, such as time constraints, autonomy, interaction, and possible lack of resources or prior knowledge. FOIL is one of the most popular and simple algorithm that generate logic definitions of classes given positive and negative examples, and a formalisation based on a DL formalism is presented in [126]. Also, other algorithms have been used to learn the definition of semantic classes, such as the one presented in [77] as well as the work proposed in [57], where terminological decision trees have been used. Moreover, the DL-learner framework has been proposed in [29], which relies on supervised learning algorithms to build classifiers that are structured in an ontology. The framework uses inductive programming paradigms to implement a service that interfaces, through an OWL representation, the core which relies on data-driven techniques and external clients. Given some prior knowledge, the framework addresses the problem to derive logic conclusions, *i.e*. deductive reasoning, and to find general statements, *i.e*. inductive reasoning. While it seems that the system could solve the first problem more easily than the second one, the number of observations required to learn how to solve both problems might not be suitable for HRI applications. Besides, the models they generate might be not intuitive, and they require a refinement process made by expert persons.

Among all the above type of knowledge representations, we are not aware of robots that can learn on-line abstract and contextualised uncertain knowledge sharable with users. Nevertheless, similar issues have been investigating for other domains, *e.g*. in the semantic web, with the purpose of learning and expressibility. For instance, in [172] a probabilistic OWL reasoner is presented, which is suitable for learning structured knowledge in the semantic web domain. Some robotic architectures rely on deterministic ontology to represent certain knowl-

edge, and such a crisp representation limits the possibility of learning new data from sensory data. In other systems, fuzzy representations have been used as well, for instance in order to deal with uncertainty perception and natural language [149]. Also, fuzzy ontology has been used to represent quantitative human activities [50]. FuzzyDL, a fuzzy reasoner for OWL ontology that supports concepts as satisfiability and subsumption, along with other features, has been presented in [22], for reasoning on OWL axioms annotated with a fuzzy degree of truth. FuzzyDL has been used in different domains, including planning [52] in a multi-robot scenario, and it has been exploited within the DL-learner framework [78] and coupled with a deep neural network [38]. Also, through fuzzyDL two extensions of the FOIL algorithms have been proposed in [1] and [98].

8.2.1 Background

SIT is an algorithm coupled with an ontology that we designed to perform one-shot structured learning of categories through demonstrations. In particular, it uses a DL formalism to reason on the similarity among categories of scenes that the robot experienced. Given a new scene, SIT exploits the Pellet OWL reasoner to classify it and returns a structure with the categories of similar scenes that the robot had seen. In particular, SIT returns a directed acyclic graph of categories, where each parent of a node classifies only a part of the perceived scene, *i.e.* a sub-scene, that have been experienced in the past.

In Chapter 7, we presented the algorithm having three phases, the perceiving, classifying, and learning phases. For demonstration purposes, those phases are called in a way that the algorithm provides an output graph for any possible scenes. In other words, SIT tries to classify an input scene and, if not suitable categories are available in the ontology, it uses the learning phase for generating a new category, from the current scene and re-performs the classification phase. Remarkably, after a category is learned from a demonstration, the algorithm looks also for a connection between the new category and the previous experiences. Although in this paper we consider the same evaluation of the phases of the algorithm as in Chapter 7, other applications might rely on a different execution of the SIT phases. In particular, in Chapter 10 we present an architecture that uses SIT for implementing memory with some cognitive aspects, such as the representation of a scene in episodic and semantic memory, as well as the evaluation of consolidating and forgetting operations.

8.2.2 Contribution

The SIT algorithm was formalised through a crisp DL and this leads to some limitations. In particular, in Chapter 7 we assumed that the in-

puts to the algorithm are perfectly representing the environment. Since in a crisp ontology symbols can have discontinuous states, a small changes in the input will generate considerably different symbols that would make the algorithm reasoning on scenes that are not consistent with the environment. In other words, the crisp implementation of SIT is not robust.

In this paper, we want to use fuzzy ontologies in order to increase the robustness of the algorithm. In particular, we want an algorithm that reason on experiences represented through uncertain knowledge, and we considered to have two different sources of uncertainties. On the one hand, the perception might be affected by noise, while on the other, the qualitative features of a scene could have some degree of truth. For instance, if the salient element of a scene is the shape of the objects on a table, they might be perceived as having a different type based on a confidence value, *e.g.* a bottle could be for some extends a cylinder, but it could be also perceived as a cone.

The algorithm generates a structure of categories that is computed as an implication graph among classes in an ontology. In a crips ontology, SIT builds an experience graph where edge relates only scene and sub-scenes, but not their possible overlapping. Our objective is to investigate the outcomes of a fuzzy formalisation of SIT in order to reason on experience overlapping. In particular, in the paper, we evaluate experience graphs where each edge has a fuzzy degree of truth, in contrast with the crisp implementation where the scene categories have been limited to exist or not exist in the representation.

Remarkably, in this paper, we present a fuzzy extension of the algorithm that behaves as the crisp SIT, if all the degrees of truth are either 0 or 1. Moreover, a fuzzy ontology allows maintaining a representation having a degree of the classification of a scene which, together with the similarity value and the degree assigned to edges of the experience graph, improves the expressibility of scene categories represented in the ontology over demonstrations.

SIT Algorithm Primer

[...][8.1]

8.1 the section summarises the contents of Chapter 7 as shown in Figure 7.2.

8.3 Problem Statement

In order to formalise a fuzzy version of the SIT algorithm, we consider a fuzzy membership value $p_z \in [0, 1]$ representing the degree of truth a particular characteristic of the environment, *i.e.* a tuple $\langle(\mathbf{r}_z, g_x, g_y), p_z\rangle$. The crisp implementation of SIT requires each instance involved in the characteristic of a fact to have a non-ambiguous type, *e.g.* $g_x \in T_x$, but this cannot be considered anymore in a fuzzy domain. Indeed, with fuzzy values we want to represent that an element has some

degree of membership in different sets, for instance, the shape of a bottle g_x could be described as a cylinder with fuzzy degree p_{x1}=.7, and a cone with membership p_{x2}=.5. Since in such a representation the concept of having a non-ambiguous type for each instance cannot be considered anymore, we need to extend the semantic interface of SIT considering all the possible types. Given a new definition of the semantic interface, in the later section of the paper, we want to adapt the perceiving, classifying, learning and structuring functions to be used with the fuzzyDL reasoner.

In particular, we assume each instance g_i involved in a characteristic $\mathbf{r}_z$ to have some membership value with each possible type in T, which are given as prior knowledge. Formally, we can rewrite the semantic interface of SIT for a fuzzy domain as

fuzzification of Equation 7.5.

$$
\begin{aligned}
f_i = \langle (\mathbf{r}_z, g_x, g_y), p_{iz} \rangle \in F : \mathbb{F} \iff \\
\langle (g_x \in T_s), p_{xs} \rangle \ \forall s \in [1, w], \\
\langle (g_y \in T_h), p_{yh} \rangle \ \forall h \in [1, w], \\
\mathbf{r_z} \in R,\ T_s, T_h \in T,\ p_{iz}, p_{xs}, p_{yh} \in [0, 1],
\end{aligned}
\tag{8.1}
$$

where, for each characteristic $\mathbf{r}_z$ involving the xy-th pair of instances with the p_{iz} degree, we assume to have also two sets of w membership values representing the types of the instances, *i.e.* $P_x = \{p_{x1}, p_{x2}, \ldots, p_{xs}, \ldots, p_{xw}\}$ and $P_y = \{p_{y1}, p_{y2}, \ldots, p_{yh}, \ldots, p_{yw}\}$. To simplify the notation, we define a fuzzy fact as a tuple $f_i = \langle D, P \rangle$, where D contains only the representation of the symbols involved, while P contains their fuzzy degree. Formally, for the i-th fact, D contains a characteristic $\langle \mathbf{r}_z, g_x, g_y \rangle$, and the types for its instances, *i.e.* $\{T_s,\ s{\in}[1, w]\}$ and $\{T_h,\ h{\in}[1, w]\}$. Instead, P contains only the fuzzy degrees associated to the elements of D maintaining the correlation between the z, x, s, y, and h indexes, *i.e.* $P = \{p_{iz}, P_x, P_y\}$.

8.3.1 Fuzzy Cardinality Restrictions

In this paper, we want to use the fuzzyDL OWL reasoner for representing the knowledge to be elaborated with the SIT algorithm. As we mentioned, the algorithm exploits the reasoner to structure experiences based on minimal cardinality restrictions, but unfortunately, fuzzyDL does support cardinality-based reasoning. This because defining the concept of cardinality is not trivial as in crisp representations, where it is enough to count the number of axioms [23]. In a crisp domain each relation represents the unit to count, but if fuzzy degrees are given, the counting operation becomes ill-posed. For instance, it is not clear if an element having a very low membership value in a set should be considered as one during the computation of the cardinality of such a set.

Several definitions have been proposed for identifying expressive counting functions, but in the literature there is not an explicit agreement on a definition [35]. Among all, the σ-count approach is mostly used, and it defines the cardinality of a set as the sum of the membership values of all its elements. This measure is simple and identifies a cardinality as a real positive number instead of a not natural numbers, *i.e.* is not a crisp value. However, it has some drawbacks. Indeed it computes the energy of a set [45] that is considered to be a measure of the cardinality since we assume that more the energy of a set, more the elements it should have. Nevertheless, this might be not always true, and the σ-count would not discriminate any differences in cardinality between a set having 10 elements with degree .1, or only one element with degree 1. Although we should evaluate the behaviour of different fuzzy cardinality definition for a specific application before to deploy our algorithm, we consider the problem of defining an expressive function for computing the cardinality of a fuzzy set out of the scope of this paper. In the following sections, we rely on σ-count to compute the cardinality, but other approaches that compute a positive number could be adopted in our algorithm.

As far as fuzzyDL is concerned, there is no definition of a minimal cardinality restriction operator, but with the σ-count we can obtain a number representing the cardinality of a class. This implies that we cannot relay on OWL reasoning directly for implementing a fuzzy version of the SIT algorithm, but we need to represent the axioms for reasoning on cardinality restrictions explicitly. Therefore, we cannot rely on a representation of categories features as $\mathbf{R}_{zx}.\Gamma_y$, because we need to represent such a description with respect to a real number, and not a class Γ_y. For doing so, we need to reconsider the SIT beliefs as a data property, *i.e.* a property between an instance ϵ and a real number, and not as a property between two instances, *i.e.* ϵ and γ_y.

8.4 The Fuzzy SIT Algorithm

In this section, we detail the definition of all the functions involved in the SIT algorithm applied to fuzzy knowledge representation. For clarity, we present the computation performed by the algorithm through a guideline example in a simple spatial scenario, but as mentioned the algorithm could be applied with different semantic interfaces as long as they assure input facts represented as in Equation 8.1.

The interface we consider for the example used in this section defines a single spatial catachrestic between objects, *i.e.* $\mathscr{N}(\mathbf{r}_1) = \texttt{front}$, where $\mathscr{N}$ is a function that maps a symbol in the ontology in its IRI identifier, *i.e.* a *name*. In our examples, instances of facts are objects, and their type could be $\mathscr{N}(\Gamma_1) = \texttt{GLASS}$ and $\mathscr{N}(\Gamma_2) = \texttt{CUP}$.

8.4.1 Input Facts

In the fuzzy representation of SIT, we define inputs facts based on the extension of the crisp semantic interface presented in Section 8.2.2. Similar to for the crisp implementation, the representation of the facts f_i are straightforward in a fuzzy DL formalism if we represent instances as individuals, characteristic as properties, and types as classes. Therefore, we can write Equation 8.1 as

fuzzification of Equation 7.6.

$$f_i = \Big\{ \langle \mathbf{R}_z(\gamma_x, \gamma_y), p_{iz} \rangle, \quad (8.2)$$
$$\langle \Gamma_{xs} \colon \gamma_s, p_{xs} \rangle \forall s \in [1, w],$$
$$\langle \Gamma_{yh} \colon \gamma_s, p_{yh} \rangle \forall h \in [1, w] \Big\}$$

which represents an input facts encoded in a fuzzy ontology.

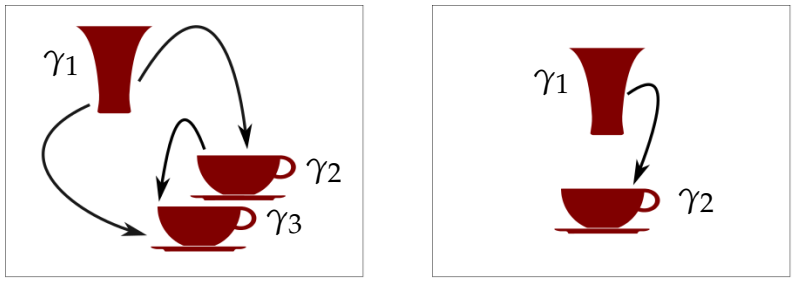
For example, Figure 8.1a can be represented in the ontology with three facts defined as

$$f_1 : \langle \texttt{front}(\gamma_1, \gamma_3), .5 \rangle, \quad (8.3)$$
$$f_2 : \langle \texttt{front}(\gamma_1, \gamma_2), .6 \rangle,$$
$$f_3 : \langle \texttt{front}(\gamma_2, \gamma_3), .9 \rangle,$$

and we assume to know in the ontology the types of all the instances, *e.g.*

$$\langle \texttt{GLASS} \colon \gamma_1, .8 \rangle, \quad (8.4)$$
$$\langle \texttt{GLASS} \colon \gamma_2, .1 \rangle, \ \langle \texttt{CUP} \colon \gamma_2, .7 \rangle,$$
$$\langle \texttt{CUP} \colon \gamma_3, .9 \rangle.$$

Equation 8.3 contains fuzzy degrees that represents the quantitativeness of the characteristic, *e.g.* p_{31}=.9 represents that the γ_3 is *mostly* in front of γ_2. Similarly, Equation 8.4 contains the fuzzy degree assigned to types representing an object, *i.e.* γ_2 could be a glass for some extent and a cup for others. Also, and more commonly in practice, the perceiving modules that identify facts of the environment could be affected by noise. In order to take into account both type of vagueness distinctly, in the guiding example of this section we consider the characteristics of a fact as quantitative fuzzy relations, while the objects types are disjointed but noisy, *e.g.* γ_2 have also been misclassified as GLASS, but with low confidence, *i.e.* p_{21}=.1. In Section 8.5.1 we will discuss more the perception of fuzzy facts particularly for our scenario, but first, we introduce all the functions involved in the algorithm. Remarkably, in the ontology we do not represent characteristics or types with a o fuzzy degree, *e.g.* in Equation 8.4 we would not state that $\langle \texttt{CUP} \colon \gamma_1, 0 \rangle$.



(a) The category Φ_1. (b) The category Φ_2.

Figure 8.1: Scenes used as guiding examples for presenting the learning and structuring bheaviour described in in sections 8.4.3 and 8.4.4.

8.4.2 Perceiving

As defined for the crisp formalisation of the algorithm, the perceiving function is in charge to compute the beliefs of a scene based on some input facts. In particular, we define a belief in a fuzzy ontology as a reified data property between the scene ϵ and a fuzzy cardinality value $c_{zsh} \in (0, \inf)$

fuzzification of Equation 7.7.

$$\begin{aligned} \mathcal{P}(F) =& \mathcal{P}(f_i) \;\; \forall i \in [1 \dots n], \\ \mathcal{P}(f_i) &= \mathscr{R}\big(\mathbf{R}_z, \mathscr{T}(\gamma_x), \mathscr{T}(\gamma_y)\big)\Big(\epsilon, c_{zsh}\Big) \quad z, x, y : f_i \\ &= \mathscr{R}\big(\mathbf{R}_z, \Gamma_s, \Gamma_h\big)\Big(\epsilon, c_{zsh}\Big) \;\; \forall s, h \in [0, w], \\ &= \mathbf{R}_{zsh}\Big(\epsilon, c_{zsh}\Big) \doteq b_r. \end{aligned} \tag{8.5}$$

Considering the guiding example of Figure 8.1a, the reification $\mathscr{R}$ of equations 8.3 and 8.4 would generate also a belief like `frontGLASS CUP`(ϵ, c_{zsh}). Remarkably, b_r is represented through a crisp data property since the number c_{zsh} is always considered to be fully related with ϵ, *i.e.* $\mathbf{R}_{zsh}(\epsilon, c_{zsh})$ is always applied with degree 1 in the ontology. Although, c_{zsh} represents the fuzzy cardinality of the beliefs b_r, the fuzzyDL reasoner would consider it only as a numerical property of ϵ. Therefore, the scene ϵ can be represented at most through m beliefs, where $m \leqslant v \cdot w^2$, *i.e.* all the zsh-th combinations spanning in the semantic interface.

To compute the fuzzy cardinality of a belief, we define first the fuzzy degree of a fact f_i through logical operations on the symbols involved in D as

$$\mathbf{R}_z(\gamma_x, \gamma_y) \;\otimes\; \Gamma_s{:}\gamma_x \;\otimes\; \Gamma_h{:}\gamma_y, \tag{8.6}$$

which defines a fuzzy membership value p_{zsh} through the t-norm between the characteristic $\mathbf{r}_z$ and the types Γ_s, Γ_h. For instance, using the Zadeh logic we can compute the fuzzy degree of a belief as

$p_{zsh} = \min\{p_{iz}, p_{xs}, p_{yh}\}$. In a general scene given through n facts, the algorithm computes $o \leqslant n \cdot w^2$ fuzzy degrees p_{zsh}, where some of those might express knowledge about the same reified combinations. Indeed, there might be facts involving symbols zxy and $z\tilde{x}\tilde{y}$ that produce through $\mathscr{R}$ the same reified characteristic zsh. For example, in equations 8.3 and 8.4 both f_1 and f_3 has some degree $p_{112} > 0$, *i.e.* in the o=5 reified facts two fuzzy degrees represent knowledge about the `front`, `GLASS`, `CUP` combination.

Let the o fuzzy degrees be divided into subsets collecting them if they represent the scene through the same zsh combination. For each elements of those subset o_{zsh}, we compute the σ-count as

$$c_{zsh} = \sum_{\mathbf{R}_z, \Gamma_s, \Gamma_h}^{o_{zsh}} p_{zhs}, \tag{8.7}$$

which represents the fuzzy cardinality of a belief b_r. The scene ϵ is represented through all the cardinalities that are possible to compute from the o fuzzy degrees, which are at most m. Therefore, differently from the crisp implementation of SIT, given n facts we do not produce in the ontology o beliefs to be matched into the m cardinality restrictions of a category during the classification phase. Instead, in the fuzzy implementation, we generate m beliefs, which already represent the facts cardinalities to be matched.

As an example consider the scene in Figure 8.1a, where input facts are deduced in equations 8.3 and 8.4. In that case $\mathcal{P}$ would generate in the ontology the beliefs

$$\begin{aligned} &\texttt{frontGLASS}\,\texttt{CUP}(\epsilon, \min\{.6,.8,.7\} + \min\{.9,.1,.9\} + \min\{.5,.8,.9\}),\\ &\texttt{frontGLASS}\,\texttt{GLASS}(\epsilon, \min\{.6,.8,.1\}),\\ &\texttt{frontCUP}\,\texttt{CUP}(\epsilon, \min\{.9,.7,.9\}), \end{aligned} \tag{8.8}$$

that defines the scene as an individual ϵ through three crisp relations that specifies the relative fuzzy cardinality. Remarkably, not all the scenes are defined through m beliefs since we do not represent the ones with cardinality $c_{zsh} = 0$.

8.4.3 Learning

The learning function $\mathcal{L}$ generates a fuzzy category, *i.e.* a class in the fuzzy ontology, given the beliefs of a scene ϵ computed from facts. In particular, $\mathcal{L}$ is supposed to evaluate the beliefs cardinality for each zsh-th reified combination, and generates a category Φ_ϵ through the definition of the conjunction of minimal cardinality restrictions represented in the ontology as fuzzy classes Σ_{k_a}.

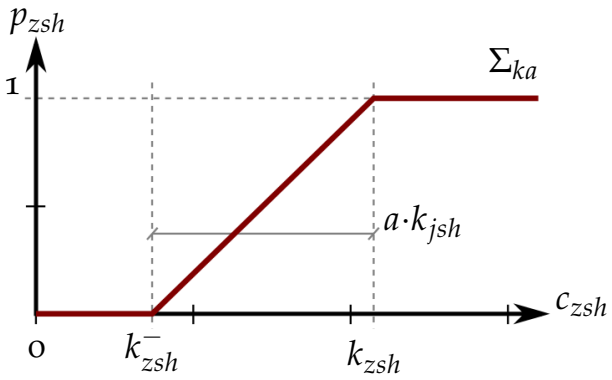


Figure 8.2: A general representation of a minimal fuzzy cardinality restriction based on a right-shoulder membership function $\Sigma_{k_a} = \mathscr{C}(k_{jsh}, a)$.

Formally we change the definition of the crisp learning function into

$$\mathcal{L}(\epsilon) = \Phi \bigotimes_{zsh:b_r}^{r\in[1\ldots m]} \mathbf{R}_{zsh}.\Sigma_{k_a} \doteq \Phi_\epsilon, \tag{8.9}$$

fuzzification of Equation 7.8.

where Φ is the root class of the experience graph, while $\Sigma_{k_a} = \mathscr{C}(k_{zsh}, a)$ is a right-shoulder function that we use to model a minimal fuzzy cardinality restriction. Differently, from the crisp implementation of SIT, a feature of a category is not a symbolic cardinally restriction, but it is the restriction of ϵ to have a property of the jsh-th belief with a numerical value, *i.e.* a cardinally c_{jsh}. Although the restriction is evaluated with a fuzzy degree p_{zsh}, it is always certain that the scene should have some cardinality for such a zsh-th combination for being of a particular category Φ_ϵ. Therefore, we encode in the fuzzy ontology the feature of a category as a crisp relation $\langle \mathbf{R}_{zsh}.\Sigma_{k_a}, 1\rangle$.

Regarding the fuzzy evaluation of cardinality restrictions, we define $\mathscr{C}$ as an operator that generates a family of functions taking two parameters, $k_{zsh} \in [0, \inf)$, which identifies the first point having a fuzzy degree equal to 1, and $a \in [0, 1]$ that is used for computing the last point having fuzzy degree equal to 0 as shown in Figure 8.2. More formally, we pose

$$k^{-}_{zsh} = k_{zsh}(1 - a), \tag{8.10}$$

where, a is a percentage of k_{zsh} that relatively defines the begging of the ramp that goes from 0 to 1 in the right-shoulder function. Remarkably, k_{zsh} would always be greater than 0, and for $a = 0$, $\mathscr{C}$ degenerates to a crisp set, while for $a = 1$ the ramp of the right shoulder would always start from the origin, *i.e.* we accept all the possible cardinalities $c_{zsh} > 0$ as respecting any restrictions, but with a low degree.

During the learning phase at the instant of time t_1, we generate specific restrictions through $\mathscr{C}$. Each restriction is stored in the ontology as a classe Σ_{k_a}, with a value $k_{zsh} = c^{t_1}_{zsh}$. During the classification phase of a later instant of time t_2, $\mathcal{P}$ might generate beliefs with a new cardinality value for the zsh-th combination, *i.e.* $c^{t_2}_{zsh}$, which will be evaluated against previously stored restrictions as we will discuss in Section 8.4.5. Instead, during the computation of the structuring function, the algorithm compares each restriction Σ_{k_a} to relate features of different categories Φ_j and Φ_l, as we will present in Section 8.4.4.

In Figure 8.2, we represent $\mathscr{C}$ as a map between cardinalities c_{zsh} and a fuzzy degree p_{zsh}. If the cardinality of a scene is $c_{zsh} \geqslant k_{zsh}$ we assume the restriction to be fully respected, *i.e.* p_{zsh}=1, instead if $c_{zsh} \leqslant k^{-}_{zsh}$ than we consider the cardinality restriction not being satisfied at all, *i.e.* p_{zsh}=0. Instead, if k^{-}_{zsh}<c_{zsh}<k_{zsh}, we use a linear function for considering the fuzzy degree of a cardinality restriction *almost* satisfied. Therefore, the parameter a identifies the relative qualitativeness of the cardinality restrictions since, if it increases, we will accept lower cardinality respecting the restriction with a degree higher than 0. However, if a decreases we would give a higher fuzzy degree to cardinality value similar to k_{zsh}, *i.e.* the referencing value that the algorithm experienced through a demonstration.

For instance, consider the belief related to the scene in Figure 8.1a and deducted in Equation 8.8. The result of $\mathcal{L}$ on such knowledge would be made of three features, which define a new category as

In the paper, we write numbers bounded in [0,1] without explicitly consider the digit before the dot, e.g. *.6. While we explicitly write 0.6 if such a quantity is not bounded.*

$$\begin{aligned}\Phi_\epsilon = \Phi \sqcap \texttt{frontGLASS}\,\texttt{CUP}.\mathscr{C}(1.2, a) \\ \sqcap \texttt{frontGLASS}\,\texttt{GLASS}.\mathscr{C}(0.1, a) \\ \sqcap \texttt{frontCUP}\,\texttt{CUP}.\mathscr{C}(0.7, a)\end{aligned} \tag{8.11}$$

Similarly for the crisp implementation, we could *read* such a representation as the statement: in the scene ϵ a GLASS has *something more* than a GLASS in front, a GLASS has *almost no* other GLASS in front, and CUP has *almost* another CUP in front. Remarkably, it could be that in a scene a CUP has *almost* 3 other CUPs in front, since the cardinality is a generic positive number, *e.g.* c_{zsh}=2.9.

8.4.4 Structuring

We want to delegate to the OWL reasoner the hard task to structure categories in an experience graph. This requires to compute the edges among all the learned categories over time and, since we define categories to be classes in an ontology, we can identify such edges as logic implications.

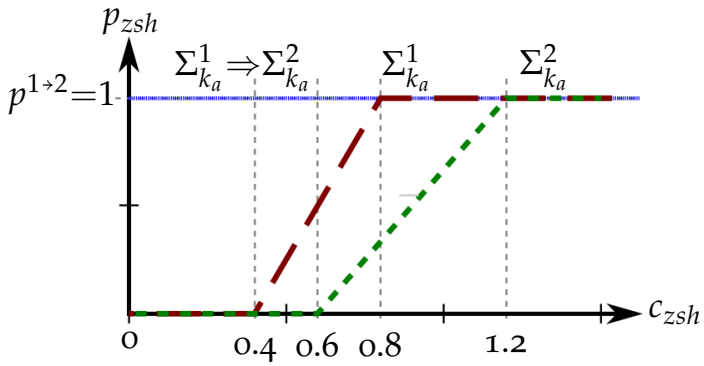
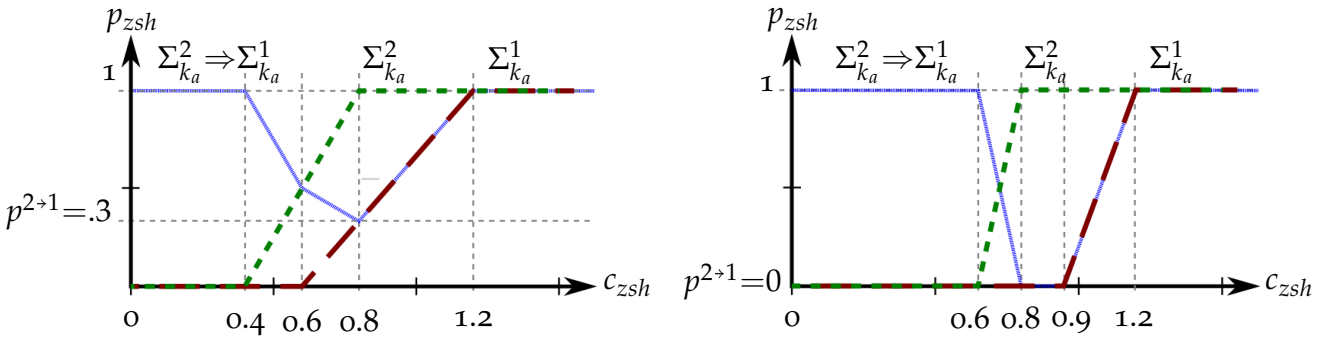


Figure 8.3: Examples of experience structuring based on fuzzy cardinality restriction computed as implication (minimum subsumption) value between a common *jsh*-th feature of two categories Φ_1 and Φ_2, *i.e.* $\langle \Phi_1 \sqsubseteq \Phi_2, p^{1\to 2} \rangle$, for a=.5. The figure are consistent with the example in Figure 8.1 where $\Sigma^1_{ka}=\mathscr{C}(0.8, a)$ and $\Sigma^2_{ka}=\mathscr{C}(1.2, a)$.

For showing how we compute implication between scene categories let consider two generic nodes of the experience graph Φ_1 and Φ_2. In accordance with Equation 8.9, each node is represented through features defined by $\mathscr{C}$ for some *zsh*-th combinations computed at some previous instant of time. If we consider the features of each category related to the same indexes *zsh*-th, and we focus on a single pair $\mathbf{R}_{zsh}.\Sigma^1_{k_a}$, $\mathbf{R}_{zsh}.\Sigma^2_{k_a}$, we can compute the fuzzy *subsumption* between the relative $\Sigma^1_{k_a}$ and $\Sigma^2_{k_a}$ restrictions, as shown in Figure 8.3. In fuzzyDL, the degree of the subsumption between two classes, *i.e.* $\Phi_1 \sqsubseteq \Phi_2$, is defined through the best entailment degree [151], which is computed as the minim value among the implication of two classes, *e.g.* the minimum of $\Phi_1 \Rightarrow \Phi_2$, as shown for graphically in the figure. We identify with $p^{1\to 2}_{zsh}$ such a value for the relevant features of the two categories. Since we consider in Equation 8.9 a category to be composed by a conjunction of features, fuzzyDL computes the degree of the subsumption $p^{1\to 2}$ as the t-norm between the subsumption of all the features of Φ_2 computed against the related *zsh*-th combinations of Φ_1, *i.e.* using the Zadeh logic $p^{1\to 2} = \min\{p^{1\to 2}_{zsh}, \forall zsh : \Phi^2\}$. In our representation, if $p^{1\to 2}$=1, it identifies that any scenes of the Φ_1 category would necessary also be of the Φ_2 category since the first always respects all the restriction of the second.

As an example, let the first category be the one derived in Equation 8.11 for the scene Figure 8.1a, *i.e.* $\Phi_1 \equiv \Phi_\epsilon$, and the second category be the one shown in Figure 8.1b. The second is as the first where we remove γ_2, and the fuzzy degree of the only remaining fact becomes $p_{z1} = .9$, *i.e.*

$$\Phi_2 = \Phi \sqcap \texttt{frontGLASS CUP}.\mathscr{C}(0.8, a). \tag{8.12}$$



(a) $\Phi_2 \Rightarrow \Phi_1$ for $a = .5$

(b) $\Phi_2 \Rightarrow \Phi_1$ for $a = .25$

Figure 8.4: Examples of experience structuring based on fuzzy cardinality restriction computed for the inverse implication shown in Figure 8.3.

Figure 8.3 shows the $\Sigma^1_{k_a}$ (in green) and $\Sigma^2_{k_a}$ (in red) restrictions for the only *zsh* feature that Φ_2 has. Also, it shows the Łukasiewicz implication $\Phi_1 \Rightarrow \Phi_2$ (in blue), which is computed for all cardinality values c_{zsh}, *i.e.*

$$p^{1\to2}_{zsh} = \min\Big\{ \min\{1,\ 1 - p^1_{zsh} + p^2_{zsh}\}\ \forall c_{zsh} \Big\}. \tag{8.13}$$

From the figure we observe $p^{1\to2}_{zsh}$=1 and, since this is the only feature that Φ_2 has we deduce $p^{1\to2}$=1, and we write $\langle \Phi_1 \sqsubset \Phi_2,\ p^{1\to2} \rangle$. This, represent that Φ_1 always fully implies Φ_2, *i.e.* if the restriction $\Sigma^1_{k_a}$ is satisfied than also $\Sigma^2_{k_a}$ would be necessary satisfied. Informally, full subsumption occurs all the times the restriction of $\Sigma^1_{k_a}$ is always greater or equal than $\Sigma^2_{k_a}$.

Contrary to a crisp ontology where two classes are equivalent *iff.* $\Phi_1 \Rightarrow \Phi_2$ and $\Phi_2 \Rightarrow \Phi_1$ are consistent axioms at the same time, in a fuzzy ontology, this is not always the case. For instance, if we compute from the example above the implication $\Phi_2 \Rightarrow \Phi_1$ and we compute the minimum subsumption as before we would obtain $\langle \Phi_2 \sqsubset \Phi_1,\ .33 \rangle$, in accordance with Figure 8.4a. This is due to the open-world assumption respected in OWL reasoning, which assume the subsumption between the second and the first category based only on the common feature. In the figure, we observe that the fuzzy degree of cardinality increases for both categories for the same values of c_{zsh}, and this generates a subsumption degree greater than 0, but smaller than 1. Remarkably, this depends for the parameter a, that in this example has been set to .5. If we consider a more conservative cardinality restriction as shown in Figure 8.4b, where the sumsumption between Φ_2 and Φ_1 is computed for the same example but with a=.25, it is possible to observe that $\langle \Phi_2 \sqsubset \Phi_1,\ 0 \rangle$. The latter would represent that a category

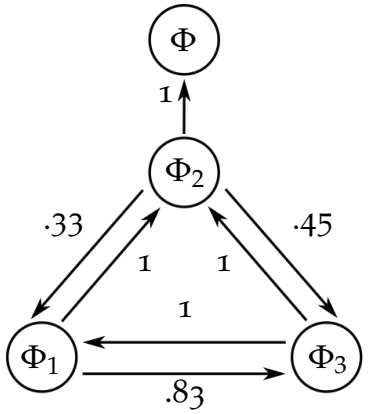


Figure 8.5: The experience graph deduced from the examples shown in Figure 8.1. Φ_3 is equivalent to Φ_1 where γ_2 is perceived as a slightly different type of object.

Φ_2 never implies Φ_1 because, if the restriction $\Sigma^2_{k_a}$ is satisfied with a degree higher than 0, $\Sigma^1_{k_a}$ might not be satisfied.

As a further example let consider a third scene and the relative learned category

$$\Phi_3 = \Phi \sqcap \texttt{frontGLASS}\,\texttt{CUP}.\mathscr{C}(1.1, a) \sqcap \texttt{frontCUP}\,\texttt{CUP}.\mathscr{C}(0.7, a) \quad (8.14)$$

which have been obtained from the example in equations 8.3 and 8.4 where we remove the axiom $\langle \texttt{GLASS}\colon \gamma_2,\ .1\rangle$; note that this scene is *similar* to the one derived in Equation 8.11. If we compute the minimum subsumption among all the possible ordered pairs of $\Phi \sqsubseteq \{\Phi_1, \Phi_2, \Phi_3\}$ we can construct an experience graph as shown in Figure 8.5, where all the categories are assumed to subsume the root Φ always with a fuzzy degree equal to 1.

In accordance with the crisp implementation of SIT we observe that Φ_2 is the simplest scene, since it does not have the object γ_2, and its features are fully matched by Φ_1 and Φ_3 which are sub-scenes, *i.e.* the implication edges in the graph with degree equal to 1. The same occurs when we evaluate $\Phi_3 \Rightarrow \Phi_1$ since we consider the scene generating Φ_3 as the one generating Φ_1 without some axioms, *i.e.* all the axioms of the latter are satisfied also in the former. Nevertheless, we observe that $\Phi_1 \Rightarrow \Phi_3$ with a high degree, since between the two categories differ only from an axiom. Similarly to an implication graph in a crisp ontology, the fact that both classes imply each other with a high degree represent that they are *almost* equivalent. The graph represents in Figure 8.5 also shows some similarities between Φ_1 and Φ_3 with Φ_2, due to a as discussed above, and identify some common features between those scenes categories as well.

8.4.5 Classifying

The classification function is in charge to compute the membership degree of a scene ϵ in a category Φ_j. This task is performed by the fuzzy reasoner that finds a match between the beliefs of a scene, represented as in Equation 8.5, and the features of a category, defined in Equation 8.9. For a *zsh* combination of the semantic interface, the former defines the cardinality about the current facts $\mathbf{R}_{zsh}(\epsilon, c_{zsh})$. While, the latter a restriction $\mathcal{C}(k_{zsh}, a)$ previously stored in a feature $\mathbf{R}_{zsh}.\Sigma_{k_a}$ of the j-th category. In this case, the reasoner would compute the fuzzy degree of c_{zsh} against the *zsh*-th left-shoulder function of Φ_j, and obtain a fuzzy degree $p_{zsh}^{\epsilon \to j}$. This is computed for all the features of the j-th category and, the final membership value of the scene ϵ in Φ_j is evaluated thought the t-norm in accordance with Equation 8.9, *i.e.* using the Zadeh logic $p^{\epsilon \to j} = \min_{zsh}^{\Phi_j}\{p_{zsh}^{\epsilon \to j}\}$. If $p^{\epsilon \to j} > 0$, we consider in the ontology that the scene ϵ is classified in the j-th category with such a degree of truth, *i.e.* the ontology contains the axiom $\langle \Phi_j : \epsilon,\ p^{\epsilon \to j} \rangle$.

For instance, let categories structured in the experience graph derived from the previous example shown in Figure 8.5, and new facts

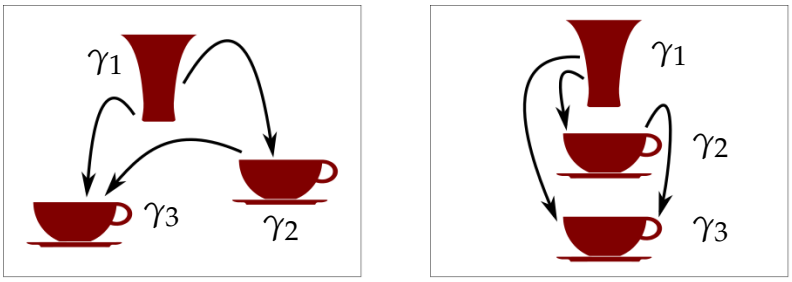
$$\begin{aligned} f_1 &: \langle \texttt{front}(\gamma_1, \gamma_3), .6 \rangle, \\ f_2 &: \langle \texttt{front}(\gamma_1, \gamma_2), .6 \rangle, \\ f_3 &: \langle \texttt{front}(\gamma_2, \gamma_3), .3 \rangle, \end{aligned} \tag{8.15}$$

where for satisfying the semantic interface we assume the type of γ_1, γ_2, and γ_3 to be as in Equation 8.4. This scene, shown in Figure 8.6a, is equivalent to the one in Figure 8.1a where γ_3 have been moved to the left-hand side of γ_1, and the degree of the spatial relation changes accordantly. In this case, by applying $\mathcal{P}$ (Equation 8.5), we obtain

$$\begin{aligned} \texttt{frontGLASS}\,\texttt{CUP}(\epsilon, 1.3), \\ \texttt{frontGLASS}\,\texttt{GLASS}(\epsilon, 0.1), \\ \texttt{frontCUP}\,\texttt{CUP}(\epsilon, 0.3), \end{aligned} \tag{8.16}$$

which defines the scene that we want to classify in the categories Φ_1, Φ_2, and Φ_3 derived in the previous section.

Let consider Φ_2 first, the most simple category in the graph. In this case, the fuzzy classification membership of the scene ϵ in the Φ_2 category depends only on one *zsh*-th combination, *i.e.* `frontGLASS CUP`; thus $p^{\epsilon \to 2} = p_{112}^{\epsilon \to 2}$. The later can be graphically computed from Figure 8.3, where to c_{112}=1.3 corresponds $p_{112}^{\epsilon \to 2}$=1. This identifies that the scene fully respects the restriction of the second category, *i.e.* $\langle \Phi_2 : \epsilon, 1 \rangle$, and we interpret this as the fact that Φ_2 models a subsection of ϵ. In the crisp formalisation of SIT, such a case would match the feature of the scene to have at least a cone in front of a glass. In the fuzzy formalisation we allow for a continuous cardinality



(a) (b)

Figure 8.6: Scenes used as guiding examples for presenting the classifying behaviour of the algorithm in Section 8.4.5.

representation and, informally, we could say that at least something less than a cone (*i.e.* 0.8) has to be in front of a glass to classify the scene as Φ_2.

Applying the same method to Φ_3 we observe that the cardinality `frontGLASSCUP`, which is restricted with k_{112}=1.1, is fully satisfied by the related belief of ϵ, *i.e.* $p^{\epsilon\to 3}_{112}$=1. This occurs similar to the previous case, because c_{112}=1.3, *i.e.* in both figures 8.1a and 8.1b a glass has some cups in front. However, when we consider the other feature of Φ_3, *i.e.* `frontCUPCUP`, which restricts the cardinally to k_{122}=0.7, we obtain a fuzzy degree of $p^{\epsilon\to 3}_{122}$=0, due to the belief c_{122}=0.3 and a=.5. Therefore, in this case, the classification of the scene is $\langle \Phi_3 : \epsilon, 0 \rangle$, since we take the minimum between the two *zsh* combinations. The same also occurs for the classification in the first category, *i.e.* $\langle \Phi_1 : \epsilon, 0 \rangle$. Therefore, in this case, the output graph of the algorithm would be a subgraph of the experience space containing the nodes Φ and Φ_2.

More generally, the result of the classification is a sub-graph Φ^* which always contains a root, *i.e.* Φ, and eventually some other nodes that we obtain through a query to the OWL reasoner. If the solution to the query is a structure Φ^* containing only the root, then the scene is considered not to be classified. Otherwise, we obtain a structure of categories containing ϵ, *i.e.* all the j-th children or successors of Φ such that $\langle \Phi_j : \epsilon, p^{\epsilon\to 3} \rangle$ and $p^{\epsilon\to 3} > 0$. In accordance with the crisp formalisation of SIT, a general sub-graph Φ^* contains classified category with features restricting a high cardinality, *i.e.* the most complex scene classified, as leaves. The parent and ancestor of the leaf categorise simpler scenes, but in contrary with the crisp implementation, in this paper, we used fuzzy cardinality restrictions that allows a not crisp definition of the complexity of a scene.

For instance, let the algorithm process a new scene shown in Figure 8.6b

$$\begin{aligned} f_1 &: \langle \texttt{front}(\gamma_1, \gamma_3), .9 \rangle, \\ f_2 &: \langle \texttt{front}(\gamma_1, \gamma_2), .9 \rangle, \\ f_3 &: \langle \texttt{front}(\gamma_2, \gamma_3), .9 \rangle, \end{aligned} \tag{8.17}$$

which, if we assume $\langle \texttt{GLASS}\colon \gamma_1,\ .8 \rangle$, $\langle \texttt{CUP}\colon \gamma_2,\ .6 \rangle$, $\langle \texttt{CUP}\colon \gamma_3,\ .9 \rangle$, is represented through beliefs as

$$\texttt{frontGLASS}\,\texttt{CUP}(\epsilon, 1.4), \texttt{frontCUP}\,\texttt{CUP}(\epsilon, 0.6), \tag{8.18}$$

Similarly to the example above we notice that $\langle \Phi_2\colon \epsilon,\ 1 \rangle$, but in this case the scene is also classified in Φ_3, *i.e.* $p^{\epsilon \to 3} = \min\{1, .8\}$. However, $\langle \Phi_1\colon \epsilon,\ 0 \rangle$ because the scene does not contain any beliefs in the $\texttt{frontGLASS}\,\texttt{GLASS}$ combination. This occurs even if Φ_1 restricts the scene to have a cardinally of only $k_{111} = 0.1$. A high classification of the ϵ in Φ_3, but a null implication of Φ_1 is a not intuitive classification since we discuss in Section 8.4.4 that in the experience graph those two categories are considered to be *similar*. Noteworthy, this occurs only for cardinality close to 0 because we would represent in the ontology any facts with a higher degree.

Although this issue also depends on the uncertainties of the incoming fact and their semantic, to solve it, we could use a method that introduces noise in the beliefs representing the scene. In particular, we rewrite Equation 8.5 such to add a belief with an artificial, small, cardinally $\bar{c}$ (*e.g.* 0.05) to all the *jsh*-th combination that are not generated from facts, *i.e.* in this way the number of beliefs of any scene would always be equal to m. In the example above, we would include in Equation 8.17 artificial beliefs as

$$\texttt{frontGLASS}\,\texttt{GLASS}(\epsilon, \bar{c}), \texttt{frontCUP}\,\texttt{GLASS}(\epsilon, \bar{c}), \tag{8.19}$$

and the scene would be classify in Φ_1 with a degree of $p^{\epsilon \to 1} = \min\{1, .8, .5\}$, with a=.5. In this way, we classify the scene in both the similar categories and the classification degree depends on $\bar{c}$ and a. Remarkably, to do not affect the learning and structuring functions, we do not process the artificial beliefs during the computation of Equation 8.9. Also, this would not increase the number of category features to reason with and improve the performances of the algorithm.

Therefore, in this example, the classification graph returned by the algorithm Φ^* contains all the nodes in Figure 8.5. For each category, the algorithm returns (*i*) its features (*i.e.* $\mathbf{R}_{zsh}.\Sigma_{k_a}$), (*ii*) incoming (or departing) edges with a fuzzy implication (or subsumption) value (*i.e.* $p^{j \to l}$), and. (*iii*) a fuzzy degree representing the membership of the scene in such a category (*i.e.* $p^{\epsilon \to j}$).

In the crisp implementation of SIT, each category also has a measure of the similarity between the feature of a category and the beliefs of a scene. This was introduced since the algorithm structures experience based on sub-scenes, which might be classified concurrently and the crisp implementation of SIT was not able to discriminate if a classified category was modelling the majority of the scene or only a small part of it. Unfortunately, such a measure for the fuzzy implementation is still an open issue due to the ambiguities that σ-count might have, as presented in Section 8.3.1. In particular, due to a, such a measure would not be bounded in [0,1], and its value might be inaccurate. Nevertheless, we argue for an algorithm that evaluates the output graph for solving such an issue when on particular semantic interfaces has been designed for a specific application.

8.4.6 Complexity

Similarly to the crisp implementation, the complexity of our algorithm depends on the size of the semantic interface, and we could order the operations from the high to the less computationally complex as the structuring, classifying, perceiving and learning. The latter becomes complex since it relies on the structuring function, but the computation of Equation 8.9 it immediate when beliefs are given. During perception, the algorithm needs to query all the instance in order to compute the cardinality of each belief, but this step could be avoided if the algorithm takes as inputs the values computed through σ-count instead of facts. However, we consider facts as input for Human-Robot interaction purposes, since we believe that for a user would be more difficult to interpret beliefs rather than facts. Moreover, during the classifying phase, the reasoner should evaluate all beliefs (at most m) against each set of restrictions of all the categories, which in the worst case are $m{\cdot}q$, where q is the number of nodes in the experience graph, *i.e.* the number of possible categories. Finally, the structuring function requires the reasoner to compare the implication of each restriction among all the possible pairs of different nodes in the graph, which in the worst case are $m(q^2 - q)$ combinations.

Among all the symbols in our representations, the number of characteristics w and types v of the semantic interface is the elements that affect most the complexity of the algorithm since in the worst case $m = v{\cdot}w^2$, due to the reification operator in Equation 8.5. This is a drawback of the algorithm since it limits the span of different facts that can be processed in the same application. Nevertheless, as we discussed in Chapter 7, the typical number of believes are considerately smaller than m in practice, and an accurate design of the characteristics of a scene might farther reduce their number at run-time, especially when the semantic interface concerns exclusive relations, *e.g.* `front` and `behind`, as well as depended relations, *e.g.* `front` and `right`.

SIT is based on cardinality restrictions that we implemented with the σ-count, which has some drawbacks as discussed in Section 8.3.1. Consequently, this might generate singularities in the representation, *i.e.* when two different scenes that are represented with the same beliefs. While for the crisp implementation of SIT this situation is infrequent and it can be avoided if the characteristics always occur with instances not of the same type, in the fuzzy implementation this phenomena might occur more frequently. We argue that this would happen more frequently especially if the semantic interface is simple, *i.e.* if more symbols might be equivalent across different facts.

It is possible to reduce m maintaining the same semantic interface, but singular representation might occur more frequently. If the application allows this drawback, we can simplify the semantics of the representation maintained by the algorithm and consider the type of only one instance of a fact. In this way, we would represent for instance that a cup has two objects in front, without specifying their type. Differently from the crisp implementation of SIT, in this case, we consider also inverse relation to not lose too much information during the computation of $\mathcal{P}$, *e.g.* in the guiding example of this section we should also consider `behind` relationships among objects. With this approach, the worst number of features would be $m{=}2v{\cdot}w$ and not quadratic anymore, *i.e.* the algorithm scales linearly with respect to the number of types that can be used for representing a scene. In particular, to simplify the algorithm we change the reified property $\mathscr{R}$ in Equation 8.5 to $\mathscr{R}(\mathbf{R}_z, \Gamma_s) = \mathbf{R}_{zs}$, and we change Equation 8.6 as

$$\bigoplus_{h\in[1,w]} \Big\{ \mathbf{R}_z(\gamma_z, \gamma_y) \otimes \Gamma_s{:}\gamma_x \otimes \Gamma_h{:}\gamma_y \Big\}, \tag{8.20}$$

which identify a fuzzy degree computed as $p_{zs} = \max\big\{\min\{p_{iz}, p_{xs}, p_{yh}\}, \forall h \in [1,w]\big\}$. Remarkably, with similar modifications, also the crisp implementation of the algorithm could be simplified in the same way.

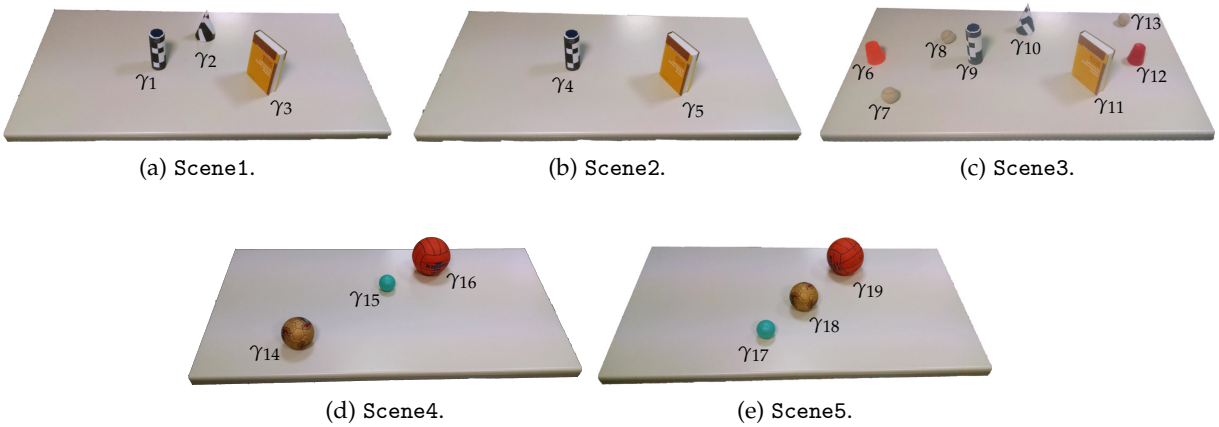
8.5 Implementation

As mentioned we based our algorithm on the fuzzyDL reasoner and the Robotic Operative System (ROS). In particular, we implemented[8.2] the algorithm in its simplified version presented in Section 8.4.6, and we tested the scenes in Figure 8.7, which have also been used to evaluate the crisp implementation of SIT[8.3]. In this paper, we present a use case we design to highlights the behaviour of the algorithm, which is discussed in the following sections. While in this section discuss some aspects related to the perception of the environment for our particular use case that influenced the algorithm.

In our use case, we consider the same scenario presented in chapters 7, 9, 10, which involves a user arranging objects on a table. For

8.2 https://github.com/EmaroLab/fuzzy_sit.

8.3 Figure 8.7 shows the same scenes of Figure 7.5.



(a) Scene1. (b) Scene2. (c) Scene3.

(d) Scene4. (e) Scene5.

Figure 8.7: Scenes used for validate the behaviour of the algorithm and compare the crisp with the fuzzy implementation.

this scenario we consider the algorithm configured with a semantic interface characterising scenes through spatial relations, *i.e.* $\mathbf{R} = \{\texttt{right}, \texttt{bheind}\}$, which are identified in a 2D space, *i.e.* the table. In this semantic interface, characteristic involves objects with with geometric types as instances, *i.e.* $\Gamma = \{\texttt{SPHERE}, \texttt{PLANE}, \texttt{CONE}, \texttt{CYLINDER}\}$.

8.5.1 Perception and Input Facts

SIT elaborates scenes at discrete instants of time, and for each of them, it assumes the ontology to contain facts in accordance with Equation 8.2. As presented in Chapter 7, we consider inputs given from the stack of ROS nodes presented in Chapter 2, which uses RANSAC simulations to segment a point cloud and estimate the shape of objects on table through the Point Cloud Library (PCL) [132]. In in Chapter 2 we propose a system that identifies a shape for each object based on the most confident RANSAC simulation. However, in this paper, we consider each object has some possible shapes with a degree of truth, in accordance with the fuzzy semantic interface of the algorithm. The latter is computed considering the rate between the points of a cluster identifying an object, and the points of such a cluster that RANSAC estimates to belong to a specific model, *i.e.* a sphere, a plane, a cone, or a cylinder. This value is one if a cluster of the point cloud has all points coherent to a geometric model, while it would be 0 if no points are belonging to such a model. We use such a measure to identify at runtime the fuzzy degree of the types of each instance γ_x and γ_y involved in a fact, *i.e.* P_x and P_y, defined in Section 8.3.

RANSAC allows to estimate also geometrical characteristics of objects, which we can use to represent them with complex shapes but, in this section, we only consider their centre of mass. Differently to

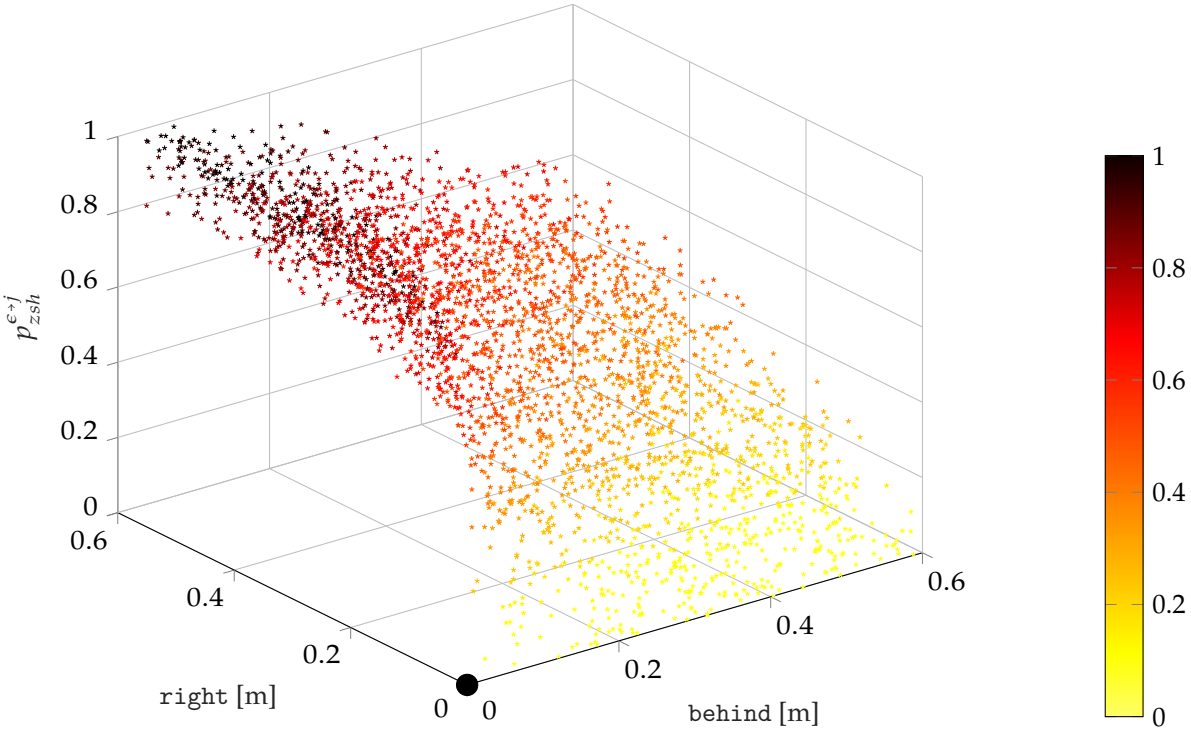
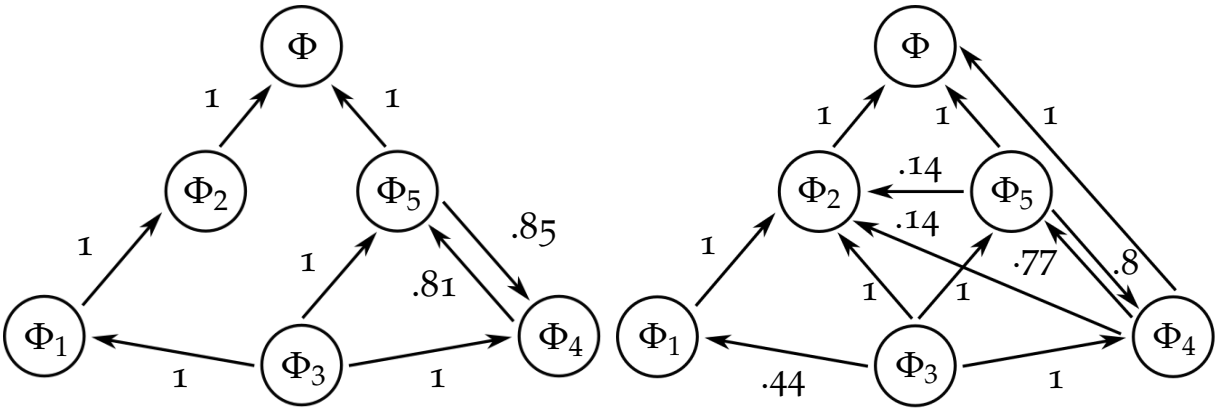


Figure 8.8: The fuzzy degree of the points on the `right`-hand side of an object, which is marked with a black dot in $(0, 0)$.

the crisp spatial relation used in Chapter 7, in this paper, we based the representation of fuzzy spatial relation on the work presented in [19]. In the latter, fuzzy morphology has been used to represent the fuzzy degree of truth of the points that are related to a referencing object of any shape. For instance, in a 2D map, this method would assign a fuzzy value greater than 0 to all the points that are on the *north* of a door, for instance. In Appendix A, we used a simplified version of that method based on a conjunction of fuzzy kernels to compute the combination of spatial relations applied between a robot and an object. In this paper, we use the same computation to evaluate the fuzzy degree of the characteristic of a fact, *i.e.* p_{iz}, and Figure 8.8 shows the measurement of the degree of truth of some points to be on the `right`-hand side of an object placed in to origin and marked with a black dot. For representing `behind` relation for the referred scenario, we merely rotate the kernel according to a reference frame shared between the robot and the user.

Furthermore, in Chapter 9 we focused our attention on the location of the reference frame that such a spatial representation requires, and we test a scenario where it was always oriented as the face and shoulders of the user that is supposed to supervise the robot. In that work, we observed that a moving point of view of the scene was confusing the user, mainly because they were trying to project themselves in a previous orientation they assumed while interacting



(a) Computed with a=.5.

(b) Computed with a=.9.

Figure 8.9: The experience graph learned from the observation of the scenes in Figure 8.7.

with the robot. The placement of a reference frame that simplified the communication is a challenging open issue for our approach, but in this paper we assumed the user to stand and face the table as long as the interaction continues, *i.e.* his or her shoulders are assumed to be aligned with an edge of the table.

Remarkably, in this scenario, the two characteristics of the semantic interface, *i.e.* `right` and `behind`, do not only depend on a reference frame, but they also dependent between each other, and this is a typical case in practice. In particular, due to the fuzzy kernels we adopted, when the degree of a point with respect to the former increases, the latter decreases, except for the points on the diagonal, where the fuzzy degree remains constant for both dimensions. Informally, in this fuzzy spatial representation, if an object has another *almost* behind, it cannot be possible that it is also *almost* in front. In accordance with the crisp implementation of SIT, and this representation affects the beliefs of a scene, and consequently the categories and the experience graph that the algorithm would generate. From a general perspective, having such a dependence between the characteristics of a scene reduce the number of combinations that beliefs of ϵ would have. This reduces consistently the possibility of the occurrence of singular representations, which have been discussed in Section 8.4.6.

8.6 Algorithm Evaluation

Figure 8.9 represent two graphs we obtained after the algorithm processed all the scenes in Figure 8.7, which are demonstrated one after the other in two different tests. Similarly to the crisp implementation, the algorithm always learned a new category, *i.e.* it never recognises a previous scene, and the result of the structuring function at the end of the demonstrations are the experience graphs shown in the figure. We observe that the fuzzy SIT consistently extends the crisp imple-

mentation since we would obtain the same experience graph shown in Chapter 7 if only 0 and 1 were used to represent input facts[8.4]. However, in this paper, we do not assume to perceive the environment perfectly.

8.4 the graphs above are the fuzzy representation of the graph in Figure 7.4b.

In particular, in all the scenes, the cone had also been represented as a cylinder with a degree between .2 and .4. Nevertheless, for all the perceived object we had a minimum fuzzy degree of the object type consistent with the environment of .7. From the structure of the experience graph in the figures, it is possible to observe two interweaving behaviours of the algorithm. One involves the scenes in figures 8.7a, 8.7b and 8.7c, where objects are introduced or removed from the table, while the other involves figures 8.7d and 8.7e, where objects have been moved on the table. We used the former example to discuss how the algorithm structures category and sub-categories of a scene, while with the second we will focus on scene classification and similarity.

Particularly for the graph in Figure 8.9a, the reasoner infers that `Scene2` is one of the most simple scenes it has never seen, therefore it is a direct child of the root. Also, it infers that all the times a scene ϵ is of the `Scene1` category with some degree greater than zero, it must necessarily have some degree also in the `Scene2` category, *i.e.* the former is a sub category of the latter with the highest degree $\langle \texttt{Scene2} \sqsubset \texttt{Scene1}, 1 \rangle$. The same also occurs for `Scene3`, since the other two categories model some of its parts. Similarly, also `Scene4` and `Scene5` are contained in `Scene3`, since in the latter three spheres are arranged as in the former from left to right, and from front to behind. Differently from the crisp implementation, in this case the `Scene4` and `Scene5` are not equivalent since the spatial relations at learning time had different fuzzy degrees.

Nevertheless, from the graph it is possible to notice that the two categories are similar since they imply each other with a high fuzzy degree. However, from the graph in Figure 8.9a it is not possible to appreciate in any differences between the relative position of the three spheres in `Scene3` and `Scene4` or `Scene5`, because in the former scene is described by much more facts than in the other two. In other words, the cardinality of the `Scene3` increased due to uncertainties of objects perceived as spheres with a low degree. This compensated the cardinality differences between the positions of the three shapes in the two scenes such to consider them to be equivalent, *i.e.* Φ_3 subsumes both Φ_4 and Φ_5 with the same degree. Informally, this shows that the algorithm tends to give more importance to similarities when the differences between the facts among the two scenes are small.

We compute the graph in Figure 8.9a through fuzzy cardinality restrictions a=.5, which represent the qualitativeness of the restriction as presented in Section 8.4.3. For this scenario, this value is suitable because it allows maintaining a consistent representation even with

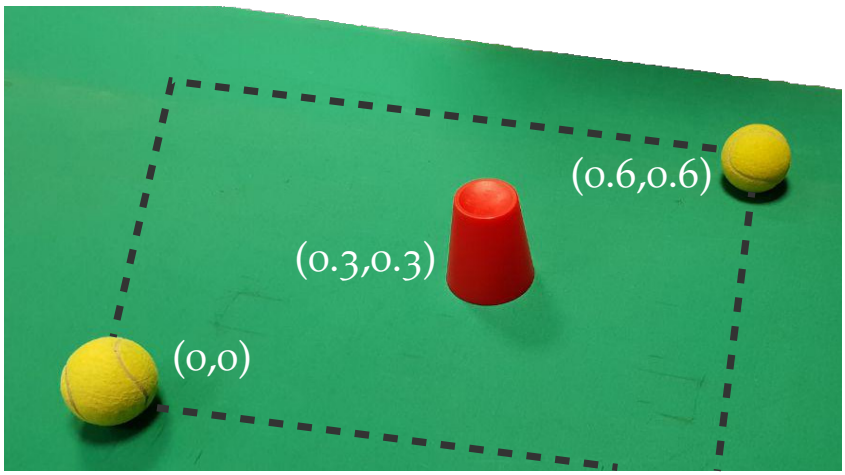
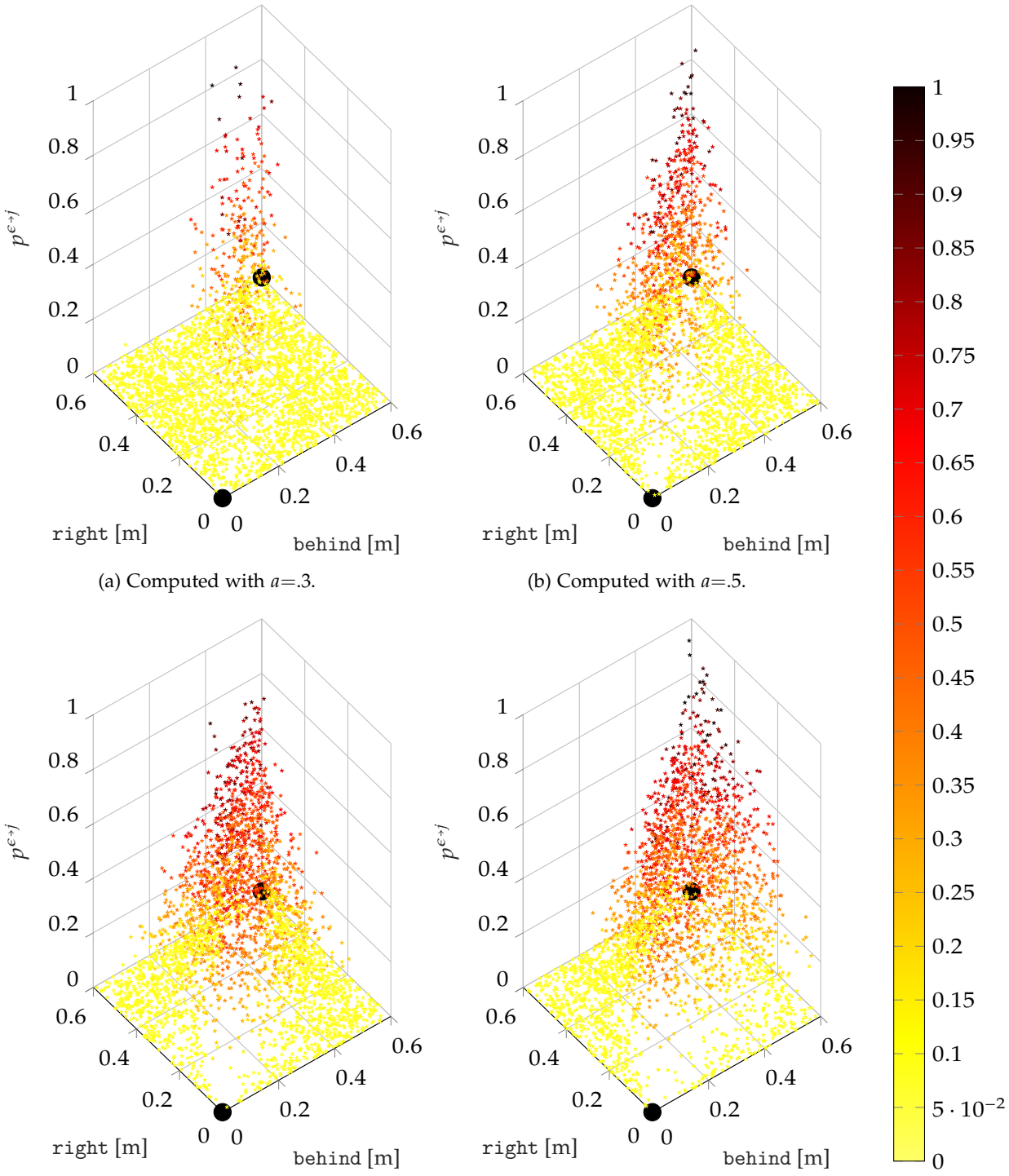


Figure 8.10: An example of learned scene used for testing the classifying behaviour if the cone is moved in the highlighted workspace (shown in Figure 8.11). In the test shown in Figure 8.11d the scene was learned with the cone in $(0.3, 0.4)$.

some noising inputs and not crisp spatial relations. However, the process of the same experiences with relaxed cardinality restrictions, *i.e.* a=.9, resulted in a more ambiguous graph, which is shown in Figure 8.9b. In this case, restrictions assigned a fuzzy degree greater than zero for lower cardinally values, and we obtained a more uncertain categorisation of the scenes. In particular, `Scene4` seems to subsume `Scene2`, even if with a low degree, because the algorithm gained more the uncertainties about the facts given at learning time. Similar, this also occurs for `Scene1` and `Scene3`, which resulted to be only partially implied.

While the experience graph contains a fuzzy degree representing implications, the output graph contains some of its nodes, which classify the scene with a fuzzy degree greater than zero. To show the classification performed by the algorithm provided with the semantic interface presented for this use case, we placed two spheres at the opposite corners of an imaginary square and a cone in the middle, as Figure 8.10 shows. Then, we used the algorithm to create a category of this scene, and we measured the classification degree of new scenes where the cone was moved on the area of the table delimited from the square. For each scene in which the cone has a different position, we compute the classification degree in the class previously learned, and Figure 8.11 shows such a value, *i.e.* $p^{e \to j}$, plotted with respect to the centre of the cone, where the two spheres are statically placed at the opposite corners of the horizontal plane, *i.e.* the table.

During this measurements, we observed noise in the degree of the identification of the object types between .1 and .23 with an average of .07 and variance of the 20.4%. While the position of objects was affected in average by $\pm$0.028m for both dimensions of the table, with a maximum value of 0.091m and a standard deviation of 14.6%.



(a) Computed with a=.3.

(b) Computed with a=.5.

(c) Computed with a=.7.

(d) Computed with a=.5.

Figure 8.11: The degree of a scene classified in the category learned from the configuration shown in Figure 8.10. The figures shows each sphere as a black dot at the corner of the workspace and, for each location of the cone on the horizontal plane of the graph, *i.e.* the table, the fuzzy classification degree of the relative scene is plotted as a point. Figures 8.11a, 8.11b and 8.11c shows the results for different values of the fuzzy cardinality restrictions parameter. Figure 8.11d, shows the results we obtained when the classified category was learned from the configuration shown in Figure 8.10, but with the cone translated on the right-hand side of 0.10m.

Figures 8.11a, 8.11b and 8.11c show the same experiment computed for a=.3, a=.5, and a=.7. When a is high, the degree of a scene very similar to the classifying category would be higher, as well as the value of slightly different scenes would be lower than in the case where a is low. In the latter case, we relax the cardinality restrictions, and the classification results to be less selective and certain.

Moreover, we can observe from Figure 8.11 that the algorithm identifies a high degree when the cone is in the centre, as it was in the scene at learning time. Also, it gives high value also for the scene in which the cone is in the diagonal that connects the two spheres. This because during learning the cone is supposed to be on the right (or left) and in front (or behind) the spheres with a theoretical degree of .5, in accordance with Figure 8.8, where an object is at the origin and the other in the centre of the workspace, which is always the case for the scene in Figure 8.10. In other words, the algorithm recognises all the scene where the cone is in the diagonal because the semantic interface we used does not consider distances between objects. Therefore all the configurations where the cone is in the diagonal are symbolically equivalent for the category learned from the scene in Figure 8.10. This is an interesting behaviour for generalising qualitative scene categories, and Figure 8.11d shows a more complex case. In particular, the latter has been obtained using the same approach as before, where the learned scene was as shown in Figure 8.10 but with the cone translated to the right-hand side of 0.10m. In this case, we can observe that the algorithm generates a smooth combination of diagonals with different inclinations between the objects since different kernels are used because the scene is not symmetric anymore.

Each graph in Figure 8.11 contains about 3700 points. During those measurements, we collected the classification time required to evaluate the scene in an experience graph having only one node. It was of 382ms with a standard deviation of 25.6%, which was measured with an Intel i5@2.5 GHz with 4 Gb RAM. Unfortunately, a systematic evaluation of the performances of the algorithm for a general case is not trivial since many factors considerably impact the computation time, such as different numbers of beliefs and combinations of shared features that the reasoner evaluates. Nonetheless, Table 8.1 shows the computation time and standard deviation we observed during the tests with scenes similar to the one shown in Figure 8.7. In particular, it shows the time spent to learn and structure a new category in an experience graph with q nodes, and even if it does not statistically characterise the algorithm for a general case, we can confirm an exponential increase of the learning time over demonstrations. In this case, the computation time is mainly demanded by the structuring function, which compared all the pairs of nodes in the graph and tends to increase exponentially with the complexity of the ontology, which is consistent with the behaviour of typical OWL reasoning tasks also for fuzzyDL [24].

q	0	1	2	3	4	5	6
time [m]	0.19	0.32	0.84	1.09	4.54	9.61	17.35
std [m]	0.10	0.52	0.26	1.05	1.27	5.61	9.43

Table 8.1: Preliminary measurements of the computation time for learning and structuring a new scene category in the experience graph. The time and standard deviation are expressed in seconds, while q is the number of nodes in the experience graph before the computations.

Although, we should analyse the temporal behaviour of the algorithm more systematically and in a more complex scenario, we observer overall improvements of the performances with respect to the crisp implementation. Probably this because the fuzzy implementation involves less symbolic and more numerical computation than in the crisp case. We believe there is a large room of improvement for performances optimisation, and we argue that this algorithm could respect soft real time requirements suitable for HRI if it relies on a well-designed semantic interface, and if the experience graph does not continuously grow over time.

8.7 Discussions

From the preliminary tests presented in the previous section, we observe that the algorithm with the simplified representation presented in Section 8.4.6 behaved consistently with the full representation that we obtained through the fuzzification of the SIT algorithm, presented in Section 8.4. In particular, the evaluated spatial semantic did not lead to singular representations, but this might occurs with other semantic interfaces. Moreover, the simplified representation might be less intuitive for a user, since the definition of the features does not explicitly contains two objects types but only one. Nevertheless, fuzzy representation improves the expressibility of the representation that the user can access through linguistic quantifiers.

In this paper, we evaluated the algorithm in a simple scenario, where we used fuzzy spatial kernels and geometric objects shape to generate the space in which the algorithm represent and reason on experiences. Nevertheless, we presented the operation performed by the algorithm for a generic semantics. In particular, the algorithm only limits the syntax in which facts should be encoded, but it does not limit their meaning, which can be subjected to further reasoning procedures. Indeed, the semantic in which the algorithm should operate is defined by the application in which the robot should be deployed and depends on its perception abilities. For instance, multimodal techniques can be used for generating facts, *e.g.* using late fusion as we presented Chapter 4. However, the algorithm is based on a symbolic formalism, and even if it can encode quantitative representation, it has been mainly designed to reason on qualitative representations of the environment.

Through the evaluation presented in the previous section, we can observe that the fuzzy implementation of SIT is robust to perception noise, as well as to uncertain relations that would characterise scenes. In particular, it consistently learns, structures, and recognises categories even if the type of the instances of the represented facts is ambiguously represented through a degrees of truth. The algorithm uses an extension on the crisp semantic interface that assumes each instance in a scene to be represented with a non-ambiguous type. Therefore, for a consistent structuring of the experience, the algorithm should reason on scenes where objects have a type but, in the fuzzy implementation, this assumption is relaxed. In other words, we assumed that instances of the environment have a prevalent type.

As far as the categorisation of sub-scenes is concerned, the fuzzy implementation of SIT fully extends the crisp formalisation. Besides, it identifies a subsumption degree, which allows to reason on the similarity between scenes differently from the crisp implementation. In particular, in this paper we presented examples of similar scenes and the importance of the learning parameter a, which is not available in the crisp implementation of SIT. Also, we proposed an algorithm to generate more expressive representations than in the crisp case. As a motivation for that, consider that all the scenes in figures 8.1a and 8.6b would be considered equivalent if a crisp `front` relation is considered. Instead, in this paper, we showed how the algorithm could reason on the fuzzy representation of the scene to classify them differently, and to reason on their similarities.

Differently from the crisp implementation, the fuzzy SIT could represent some interpretations of scene overlapping. However, this requires to design reasoning procedures based on complex graphs generated through SIT, which is challenging problem out of the scope of this paper. Another challenging issue to evaluate more in detail is the scalability of the algorithm. In particular, we discuss the worst case complexity and we preliminary report some computation times, but a more deep analysis should be performed through the comparison of different semantic interfaces. For a long time application, we argue for an experience graph that does not continuously increases over time as we presented in this paper with showing purposes. In particular, the learning phase of the algorithm should be used when required only, and the experiences should be removed if they are not needed anymore. This not only would improve the performances of the algorithm, but also its output graph would be simpler to be used from an external component, and it might contain less ambiguity as well, *e.g.*, a similar scene. In the preliminary tests presented in the previous section, we observe that the algorithm with the simplified representation presented in Section 8.4.6 behaved consistently with the full representation that we obtained through the fuzzification of the SIT algorithm, presented in Section 8.4. In particular, with the

evaluated spatial semantic we did not experience singular representations, but this might occurs with other semantic interfaces. Moreover, the simplified representation might be less intuitive for a user, since the definition of the features does not explicitly contains two objects types but only one. Nevertheless, fuzzy representation improves the expressibility of the representation that the user can access through linguistic quantifiers.

In this paper, we evaluated the algorithm in a simple scenario, where we used fuzzy spatial kernels and geometric objects shape to generate the space in which the algorithm represent and reason on experiences. Nevertheless, we presented the operation performed by the algorithm for a generic semantics. In particular, the algorithm only limits the syntax in which facts should be encoded, but it does not limit their meaning, which can be subjected to further reasoning procedures. Indeed, the semantic in which the algorithm should operate is defined by the application in which the robot should be deployed and depends on its perception abilities. However, the algorithm is based on a symbolic formalism, and even if it can encode quantitative representation, it has been mainly designed to reason on qualitative representations of the environment.

Through the evaluation presented in the previous section, we can observe that the fuzzy implementation of SIT is robust to perception noise, as well as to uncertain relations that would characterise scenes. In particular, it consistently learns, structures, and recognises categories even if the type of the instances of the represented facts is ambiguous, and their relations has degrees of truth. The algorithm uses an extension on the crisp semantic interface which assumes that each instance in a scene must be represented with a non-ambiguous type. Therefore, for a consistent structuring of the experience, the algorithm should reason on scenes where objects have a type but, in the fuzzy implementation, this assumption is relaxed. In other words, we assumed that instances of the environment have a prevalent type.

As far as the categorisation of sub-scenes is concerned, the fuzzy implementation of SIT fully extends the crisp formalisation. Besides, it identifies a subsumption degree, which allows to reason on the similarity between scenes differently from the crisp implementation. In particular, in this paper we presented examples of similar scenes and the importance of the learning parameter a, which is not available in the crisp implementation of SIT. Also, we proposed an algorithm to generate more expressive representations than in the crisp case. As a motivation for that, consider that all the scenes in figures 8.1a and 8.6b would be considered equivalent if a crisp `front` relation is considered. Instead, in this paper, we showed how the algorithm could reason on the fuzzy representation of the scene to classify them differently, and to reason on their similarities.

Differently from the crisp implementation, the fuzzy SIT could represent some interpretations of scene overlapping. However, this requires to design reasoning procedures based on complex graphs generated through SIT, which is a challenging problem out of the scope of this paper. Another challenging issue is a detailed evaluation of the algorithm scalability. In particular, we discuss the worst case complexity and we preliminary report some computation times, but a more deep analysis should be performed through the comparison of different semantic interfaces. For a long time application, we argue for an experience graph that does not continuously increases over time as we presented in this paper with showing purposes. In particular, the learning phase of the algorithm should be used when required only, and the experiences should be removed if they are not needed anymore. This not only would improve the performances of the algorithm, but also its output graph would be simpler to be used from an external component, and it might contain less ambiguity as well, *e.g.* a similar scene. In Chapter 10, we use the crisp implementation of the algorithm to tackle this issue with an architecture that encodes, stores, retrieves, consolidates and forgets experiences, and we observe that the fuzzy implementation could be used with minor changes to such an architecture.

8.8 Conclusions

The paper presented the extensions of the SIT algorithm described in Chapter 7 with a fuzzy ontology. The algorithm uses symbolic inputs retrieved from the environment at discrete time instants to learn a graph of experience in a one-shot manner. Each node of the graph is a categorisation of the observations, and they can be used for classifying future scenes. Also, the graph structure sub-scenes and allows to reason on the similarity between experiences at different instant of time.

The algorithm has been designed for HRI scenario and maintains robot's experiences through logical formalism that are understandable by a user. For interacting purposes, we design an algorithm that reasons for uncertain and qualitative representations of the environment. We deployed our algorithm in a simplified spatial scenario to present its behaviour when a fuzzy ontology is used. Contrary to the crisp implementation of the algorithm, in this paper we showed that with fuzzy reasoning we could perform structured learning in uncertain environments, *i.e.* the fuzzy implementation of SIT is robust to noise and vague descriptions of the environment.

We discussed about the limitation of this implementation of the algorithm, that are mainly due to the issue of reasoning on fuzzy cardinality restrictions, which might generate inaccurate representation. However, we argued that those case are sporadic in practice and can

be avoided by interfacing adequately the algorithm with perceiving components. Also, we presented the complexity of the algorithm, and we proposed a simpler representation of the knowledge for improving the performances.

As further work, we want to deploy the algorithm in more realistic scenarios and evaluate its scalability for long-term applications. This would generate large experience graphs, through which we can evaluate the algorithm more systematically. Also, given complex experience graphs, it could be possible to design further reasoning procedure, as well as procedures to communicate the experience of a robot to a user, which might want to correct them through dialogues.

Acknowledgements

This work has been partly supported by a grant of the Fondazione/Stiftelsen C.M. Lerici awarded to the first author in 2017.

Part V

Supervise to Consolidate and Forget

This part shows architectures which have been developed based on our design pattern.

9

Dialogue-based Supervision and Explanation of Robot Spatial Beliefs: a Software Architecture Perspective

Abstract

The paper presents a software architecture allowing a robot to learn new compositions of objects in table-top scenarios by human demonstrations. The robot qualitatively represents those scenes, reason upon their similarity, and interact with humans through dialogues to *talk about* represented scenes. We formalise the robot behaviour based on a Description Logic representation of scenes through *spatial beliefs*, *i.e.* learned logic predicates, on which the robot applies symbolic reasoning to recognise and explain the scene. We exploit the logical structure of predicates in a software architecture that enables a robot exposing its beliefs, and if required, it allows a human supervisor to apply corrections in a form akin to robot active perception.

The paper critically discusses the design of the software components and their interfaces, discriminating between knowledge representation and dialogue management. Those components are developed for human-robot knowledge sharing applications involving visual, verbal, and auditory modalities of interaction. Software components are treated as grey boxes managing an ontology-based formalisation of robot beliefs through four contextualised dialogues, for which we present a unique design pattern.

robot supervision · learning by showing · dialogue management

Authors

Luca Buoncompagni and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

Published in the Proceedings of the 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2018, Nanjing, CN).

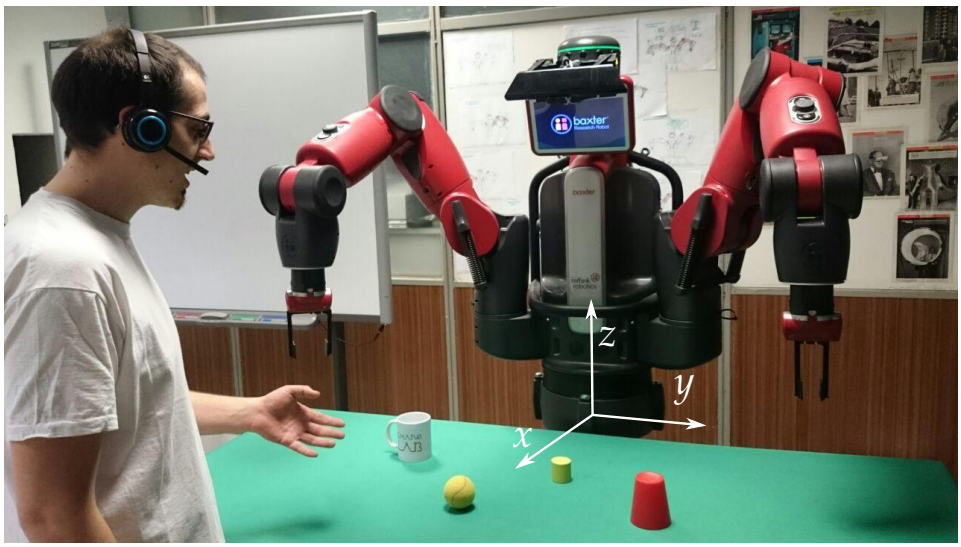


Figure 9.1: The experimental scenario.

9.1 Introduction

The Defence Advanced Research Projects Agency (DARPA) recently advocated the need for intelligent systems trading-off *explainability* and *inference accuracy* when Artificial Intelligence (AI) techniques are used [69]. It is argued that an increase in performance for such systems can be achieved when AI agents interact in synergy with humans.

AI systems should be able to *anchor* numerical (sensor-based) measurements of the environment to a symbolic representation understandable by humans [42]. In the context of Human-Robot Interaction (HRI) and Human-Robot Cooperation (HRC) scenarios, we foresee interaction processes where a person can access anchored symbols in the robot knowledge representation system with the purpose of teaching a robot how to interpret sensory data, to supervise it during interaction tasks, and to obtain information about performed inferences and reasoning processes. Due to the nature of symbolic knowledge, we argue that communication at the *verbal level* is of the utmost importance in real-world settings, although different modalities (to be integrated in a well-defined framework) should play a decisive role to achieve such an interaction flexibility, *e.g.* deictic gestures [25].

This paper addresses the design of software architectures for robots able to:

- maintain a representation of salient features of the environment acquired through multiple interaction modalities, and
- enable humans to instruct, supervise, and obtain understandable information from a robot at a symbolic level, via speech-based dialogues anchored to the robot belief and reasoning spaces.

In particular, the paper discusses three use cases in a tabletop scenario where a human and a robot face each other for interaction or cooperation purposes (Figure 9.1). The robot is a passive observer that learns, recognises, and explains scenes perceived via a RGB-D sensor, which representation is *assessed* and if necessary *amended* by the human through dialogue. Three phases are envisaged in the interaction process: first, the environment is set up according to the task at hand; second, the robot interactively performs reasoning upon scenes and learning, while being supervised by a human; third, it describes the outcome of such a process to the human. In this context, our main contribution is the definition and implementation of relevant design/interaction patterns for reusable software components, to be adopted when two specific HRI- and HRC-oriented robot functionalities are required, namely (*i*) the use of a knowledge base grounding reasoning techniques, specifically in the case of Description Logic (DL) based reasoning about perceived scenes, and (*ii*) the adoption of contextualised dialogue management, for which we present an architectural template discussed for four different types of verbal interaction.

The need for AI systems to be able to explain or at least expose with sufficient clarity their representations and inferences is considered of the utmost importance in recent scientific and technological trends. However, such a capability has non obvious implications on the interaction process, as well as on the employed representation and reasoning approaches.

Data-driven frameworks for classification and inference (*e.g.* based on hidden Markov models or Random Fields) do not employ symbolic knowledge, and their behaviour can be hardly understood by non-technically skilled people. A few systems developed for explanation purposes have been proposed, a prototypical example being the work described in [26], where a graphical approach based on Principal Component Analysis is adopted. Still, a huge amount of training is needed for a human to understand the meaning associated with data representation. In [43], visual and audio information is used for robot teaching through reinforcement learning, but no clue is given about how to access learned policies. Ideally, symbolic approaches are more prone to ground human-robot communication processes. For instance, the work in [61] shows how a symbolic representation allows humans to interact with a mobile robot that asks for clarifications about its workspace, whereas in [139] humans can instruct a robot on how to formalise STRIPS-based planning domains. A framework for accessing robot beliefs through dialogues has been described in [163]. However, no proposal for how to assess learned robot beliefs is discussed.

In software architectures for robots, a symbolic representation is usually maintained in a knowledge base encoding prior knowledge, the relevant state of the environment, as well as the robot abilities.

Interaction processes require such knowledge base to be continuously updated. Depending on their semantics, encoded symbols are processed by reasoning algorithms grounding robot behaviours. In order to sustain dialogues with humans, such reasoning algorithms can be bi-directionally mapped to appropriate dialogue structures. This mapping has been investigated to ground explainable AI systems, *e.g.* enabling robots to explain why they did not perform a certain action [62]. The Description Logic formalism is a *de facto* standard to represent knowledge, and it performs semantic consistency checks about predicative knowledge in ontologies. In [92] such a feature has been exploited to manage dialogues aimed at anchoring linguistic object features to RGB-D images.

The paper is organised as follows. Section 9.2 describes the modules used to represent information. Section 9.3 focuses on human-robot dialogue management. A discussion about three use cases is provided in Section 9.4. Conclusions follow.

9.2 Knowledge Representation and Robot Beliefs

9.2.1 Overview

Figure 9.2 shows the proposed interaction patterns from a Component-Based Software Engineering (CBSE) perspective, graphically shown through structural and behavioural Unified Modelling Language (UML) diagrams. In the Figure, boxes represent software *components*, which provide and require *interfaces* to operate. The latter are graphically identified with the ball-socket notation. All interfaces are characterised by a specific *message* defining the data type exchanged among components through *ports* (*i.e.* data communication channels). Ports are indicated with small component rectangles, and enumerated with $\mathcal{P}$ symbols.

We specify the proposed components for the widely adopted Robot Operating System (ROS) framework. Therefore, we assume a component to be a *node* or a *service*, the former characterised by a *spinning rate*, while the latter providing services on demand. In the first case, all messages are exchanged using the *Publish-Subscribe* policy, *e.g.* through $\mathcal{P}_2$, $\mathcal{P}_4$, or $\mathcal{P}_8$, whereas in the second case a service is provided with at least a port with a required and a provided interface, *e.g.* $\mathcal{P}_{13}$, $\mathcal{P}_{24}$, or $\mathcal{P}_{26}$, respectively, whose messages follow the *Request-Process-Reply* pattern (in our case, assuming *blocking* calls). Therefore, nodes are assumed to communicate asynchronously, whereas services operate synchronously. It is noteworthy that we adopt the *meta* node (or service) UML formalism to indicate components that are internally constituted by other nodes and services.

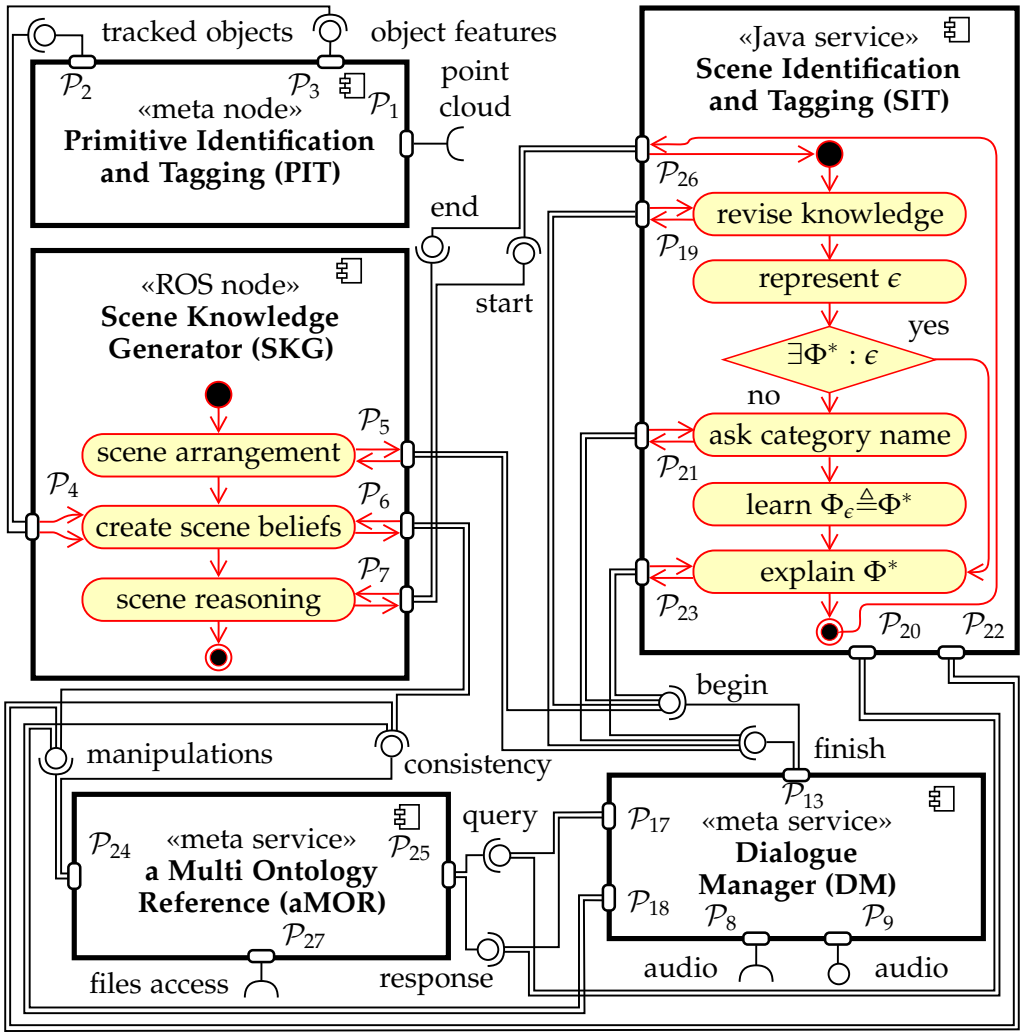
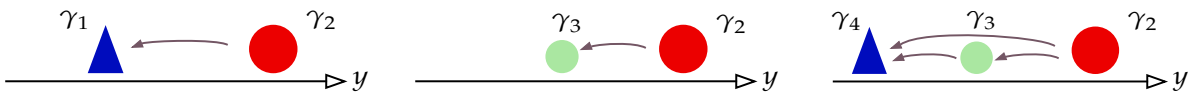


Figure 9.2: Components and interfaces.

9.2.2 Primitive Identification and Tagging

Figure 9.2 on the top-left shows the Primitive Identification and Tagging (PIT) meta-component, which has been thoroughly described in Chapter 2. PIT implements a stack of ROS-based nodes making use of the Point Cloud Library (PCL) [132] to acquire RGB-D data and the Random Sample Consensus (RANSAC) estimation algorithm to classify object shapes in the robot workspace. PIT segments objects located on a table, identifies their salient geometrical features, and assesses their mutual *qualitative* spatial relations (first column of Table 9.1[9.1]). PIT orchestrates two computational strands: on the one hand, object *position tracking*, which defines unique object identifiers in the form $\texttt{ID}_i$ for estimating qualitative spatial relations based on their centre of mass; on the other hand, the *estimation of object features*, such as their colour, shape, and geometrical coefficients. It is noteworthy that the two processes have different computational requirements, in terms of

9.1 computed as shown in Section 7.6.4.



(a) Scene 1. (b) Scene 2. (c) Scene 3.

Figure 9.3: Three examples of scenes experienced by the robot for the presented use cases; y represents the `right` direction.

node spin rates. As a consequence, PIT provides two asynchronous interfaces connected from $\mathcal{P}_2$ and $\mathcal{P}_3$, respectively, to port $\mathcal{P}_4$ of the Scene Knowledge Generator (SKG) meta-component.

From a computational perspective, we anchor symbols γ_i to real objects through their unique identifiers. However, such a choice is not appropriate to ground dialogues, specifically when different instances of the same object category must be disambiguated in the same scene as perceived by the robot, which can be done using their *most relevant* features. As shown in Table 9.1, we order each object features from the most (*e.g.* the *shape*) to the less important (*i.e.* the `ID`), and we choose the first m features allowing for an unambiguous anchoring between real-world objects and their symbolic representation. For instance, Figure 9.3c shows an example where objects are anchored as *red sphere* (γ_2), *green sphere* (γ_3), and *cone* (γ_4). While the *shape* feature can discriminate between *sphere* and *cone*, the two spheres can be disambiguated only by their *colour* feature. In order to design scene recognition and learning approaches, such anchoring policy results to be too restrictive, since it implicitly suffers from a combinatorial explosion on the number of object instances, and therefore to those aims we consider only geometrical shapes, *i.e.* *shape* is the only object's feature used for describing spatial relations among objects, as shown in the second and third columns of Table 9.1. It results that as far as spatial relations are concerned, all *sphere* objects are equivalent, but this is not valid when a symbol γ_i is used to ground dialogues, where all other object features are used.

The associated reasoner recognises, learns and explains tabletop scenes based on logical predicates, as shown for illustration in the first column of Table 9.1. Those predicates are represented in an ontology describing spatial relations between objects, and initialised using the information provided by PIT. We consider an *a priori* defined set of possible features and spatial relations, respectively *sphere*, *cone*, and *cylinder*, as well as *on the right of*, *behind*, and the inverse relations. For instance, in Figure 9.3a robot spatial beliefs are represented in the form `right(`γ_1`:CONE,`γ_2`:SPHERE)`, which can be expressed using such verbal-level sentences as "γ_1 is a cone", "γ_2 is a sphere", and "The cone has the sphere on the right hand side", with respect to the global reference frame as seen by the supervisor (Figure 9.1).

Object features and spatial relations	Scene representation	Scene categories
γ_1 : CONE, blue, y_1, ID_1; γ_2: SPHERE, red, y_2, ID_2; right(γ_1,γ_2).	S : rightCONE(ϵ,γ_2).	$\Phi_1 \equiv \Phi \sqcap$ $\geqslant 1$ rightCONE.SPHERE
γ_2 : SPHERE, red, y_2, ID_2; γ_3: SPHERE, green, y_3, ID_3; right(γ_3,γ_2).	S : rightSPHERE(ϵ,γ_2).	$\Phi_2 \equiv \Phi \sqcap$ $\geqslant 1$ rightSPHERE.SPHERE
γ_2: SPHERE, red, y_2, ID_2; γ_3: SPHERE, green, y_3, ID_3; γ_4: CONE, blue, y_4, ID_4; right(γ_4,γ_3), right(γ_4,γ_2), right(γ_3,γ_2).	S : rightCONE(ϵ,γ_3), rightCONE(ϵ,γ_2), rightSPHERE(ϵ,γ_2).	$\Phi_3 \equiv \Phi \sqcap$ $\geqslant 2$ rightCONE.SPHERE $\sqcap$ $\geqslant 1$ rightSPHERE.SPHERE

Table 9.1: Predicates generated in the ontology for the three scenes in Figure 9.3 learned in sequence.

9.2.3 Scene Knowledge Generator

SKG is involved in the first phase of the interaction outlined in the Introduction, and it is designed as a node calling three services in sequence (therefore exploiting the *Computational* design pattern) while evolving from an initial to a final state as shown in Figure 9.2. SKG serves the purpose of allowing a human to arrange the tabletop environment, while enabling the robot to observe the table using RGB-D data as managed by PIT. Remarkably, we assume that the scene the robot perceives is not modified during the other two interaction phases.

SKG requires the Dialogue Manager (DM) service (with a signal from $\mathcal{P}_5$ to $\mathcal{P}_{13}$) to notify the human supervisor that the robot is observing the scene. Human supervisors can move the objects, and then trigger the scene reasoning process using specific sentences, which generates a message from $\mathcal{P}_{13}$ to $\mathcal{P}_5$. From $\mathcal{P}_6$, which is connected to a Multi Ontology Reference (AMOR)[9.2] service, the most up-to-date robot spatial beliefs from RGB-D data (provided using $\mathcal{P}_4$) are mapped into predicates in the ontology, and then the SIT service is called through $\mathcal{P}_7$. Finally, SKG terminates, and the sequence can be performed again, thereby reasoning on a new scene.

9.2 presented in Section 6.2.1

9.2.4 Scene Identification and Tagging

The Scene Identification and Tagging (SIT) service is involved during the second and the third phases identified above. Through $\mathcal{P}_{26}$, SIT is the fundamental module enabling scene representation and explanation. In the second interaction phase, after a scene has been acquired and modelled at the symbolic level, a human supervisor can

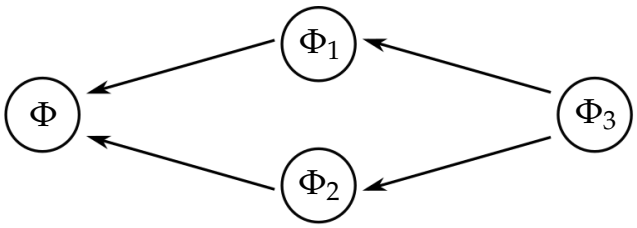


Figure 9.4: The scene implication graph learned based on the knowledge in Table 9.1.

assess robot beliefs before scene reasoning occurs, while in the third phase a consistency check about robot explanations related to the reasoning process is performed. Scene reasoning employs a simplified version of the method proposed in Chapter 8, which exploits Fuzzy DL reasoning. In this case, we consider a Boolean implementation[9.3] based on the Pellet OWL reasoner [144]. Such a DL reasoner structures the symbolic knowledge as shown in Table 9.1, where object features and the related spatial relations in the first column are mapped to the representation shown in the other two columns. In particular, in the second column of the Table, each row shows a scene description ϵ characterised by predicates in the form $\texttt{rightCONE}(\epsilon, \gamma_2 : \texttt{SPHERE})$, which semantics can be expressed as "In the scene ϵ a cone has a sphere on the right hand side". Using the symbolic representation, DL-based *instance checking* is enough to classify the symbol ϵ within a set of scene categories Φ_i, where each category may be learned via demonstrations.

9.3 presented in Chapter 7.

As it can be seen in Figure 9.2, SIT requires the human assessment of the representation provided by SKG, which is used to generate scene instances ϵ in the ontology. SIT performs instance checking to classify a scene ϵ as part of one or more scene categories Φ_i[9.4]. If this does not occur, *i.e.* $\forall \Phi_i \sqsubset \Phi$ it happens that $\Phi_i(\epsilon)$ is inconsistent, a new scene category $\Phi_\epsilon \sqsubset \Phi$ is *learned* using ϵ as a template, as shown in the third column of the Table. Therefore, while ϵ is a unique instance that may change every SKG iteration, Φ is a structure in the ontology that is populated via a set of demonstrations (Figure 9.4). It is noteworthy that by definition Φ_ϵ is always the best category for the scene ϵ from which it has been learned, *i.e.* for the same row of Table 9.1, the scene in the second column always fits perfectly with the scene category restrictions in the third column. As a consequence, the robot can describe the scene based on the classification of ϵ in the scene categories in Φ, for both branches of the *if* block in the SIT meta-component of Figure 9.2.

9.4 the set of scene categories is referred as $\Phi = \{\Phi_i; \prec\}$ as in Chapter 7.

SIT learns a scene category from demonstration by counting the number of spatial relations of ϵ and limiting their cardinality in the definition of Φ_ϵ. Such definitions are made up of predicates humans can understand, *e.g.* the logic formula $\Phi_i = \geqslant 1 \texttt{rightCONE.SPHERE}$ in

Figure 9.3a can be easily interpreted as "In Φ_i at least one cone has a sphere on the right hand side".

We exploit DL-based classification to ascertain scene similarities expressed as *implications* between scene categories. For instance, after the robot learns all the scene categories related to actual scenes shown by humans, an *implication* graph is induced in the ontology representation. The implication graph related to the scenes we consider here is shown in Figure 9.4. In this graph-like representation, nodes are scene categories, and oriented edges are logical implications between them, *e.g.* for the scenes in Figure 9.3b and Figure 9.3c, the DL reasoner infers that "Φ_3 includes Φ_2". Such a logical implication is consistent because all the cardinality restrictions of Φ_2 are also respected in Φ_3. In fact, $\Phi_2(\epsilon)$ is consistent if $\Phi_3(\epsilon)$ holds, but not *vice versa*, *i.e.* $\Phi_3 \sqsubset \Phi_2$ in a DL-induced semantics. It is interesting to note that Φ implicitly represents a scene that includes all the demonstrations given to the robot so far. It is possible to refer with Φ_* as the *best* classification category for a scene instance ϵ, as the node of the graph classifying ϵ that is farthest (*i.e.* the most specific one) with respect to Φ.

In order to achieve such a behaviour, SIT calls DM services to start three different types of dialogue with the human supervisor. In the first dialogue type, the predicates generated from RGB-D data and represented in the knowledge base are inspected ($\mathcal{P}_{19}$), while the second dialogue type occurs if the scene ϵ is not recognised ($\mathcal{P}_{21}$) for giving a meaningful identifier to the new scene category Φ_ϵ, *i.e.* an anchoring name. In the third dialogue type, the robot describes the scene in so far as scene category definitions and the implication graph are concerned ($\mathcal{P}_{23}$).

SIT also exploits the services provided by AMOR ($\mathcal{P}_{20}$, $\mathcal{P}_{22}$) to update and reason upon scene-related predicates. It is important to remark that since we adopt Boolean predicates, uncertainty and vagueness possibly associated with spatial relations are not considered. Indeed, since RGB-D data are affected by noise, the corresponding spatial relations in Table 9.1 could slightly differ over time, and in general the sensitivity of the overall system behaviour on sensor noise can be not negligible. While the interaction process unfolds, if the human supervisor assessment does not correct wrong predicates, scene explanation can possibly be inconsistent with the real situation, *i.e.* symbol anchoring is wrong. However, this is not generically true for a Fuzzy DL implementation, which may exploit the representation of vague concepts to cope with and limit anchoring errors. We presented such a simplified spatial representation, that requires knowledge revision, while focusing on the software designing of an architecture allowing for such a revision process.

9.2.5 A Multi Ontology References

A general description of the a Multi Ontology References (AMOR), and its connection with ROS-based nodes or services (ARMOR), can be found in Chapter 6, in particular as a service to use the Ontology Web Language (OWL) [73] with robot-based architectures.

The management of a symbolic representation in robot architectures leads to a number of computational issues, because many relevant traits of the robot workspace are out of the robot control and should be dynamically updated at the representation level. In principle, many components of reactive/deliberative architectures can generate robot beliefs to be stored in the symbolic representation. All such robot beliefs may be used by reasoners or even by other components to ground actual robot behaviours. However, only robot beliefs actually used to that aim should be explicitly represented, because reasoning processes are time-consuming tasks which complexity also depends on the number of such beliefs. As a consequence, software components generating data to be represented in the knowledge base are subject to synchronisation, because employing a DL reasoner to update the knowledge base when a single belief is modified is not computationally feasible in practice. In order to address these issues, AMOR exposes two services to maintain and access robot beliefs in a thread-safe manner, as well as whole shared semantics for all software components.

As it can be seen in Figure 9.2, AMOR provides a *manipulation* ($\mathcal{P}_{24}$) and a *query* ($\mathcal{P}_{25}$) service, as well as an interface to trigger knowledge base loading and storage ($\mathcal{P}_{27}$). These two services use a ROS-Java hierarchy of OWL-based operations specified through directives (*e.g.* `add`, `remove`, `classify`, `imply`, etc.) involving properties and data, or sets and symbols (*i.e.* predicates in our case). In particular, the first service replies with a consistency signal (*i.e. successful* manipulation), while the second service provides the caller with the list of queried predicates. It is noteworthy that such services might introduce delays, but in our work those respect soft real time constraints as far as dialogues are concerned. A user experience and system reactivity assessment is out of the scope of this paper.

9.3 Human-Robot Dialogue Management

9.3.1 Dialogue Manager

In our scenario, four different dialogue types are used (Figure 9.5), which are based on two components, namely a Speech Interaction Manager (SIM) and a SIM-driver, *i.e.* one of the components on the right-hand side of the Figure. Each SIM-driver implements a service that bootstraps a dialogue and is notified when the dialogue terminates, using Boolean signals streamed on $\mathcal{P}_{13}$. The component controls

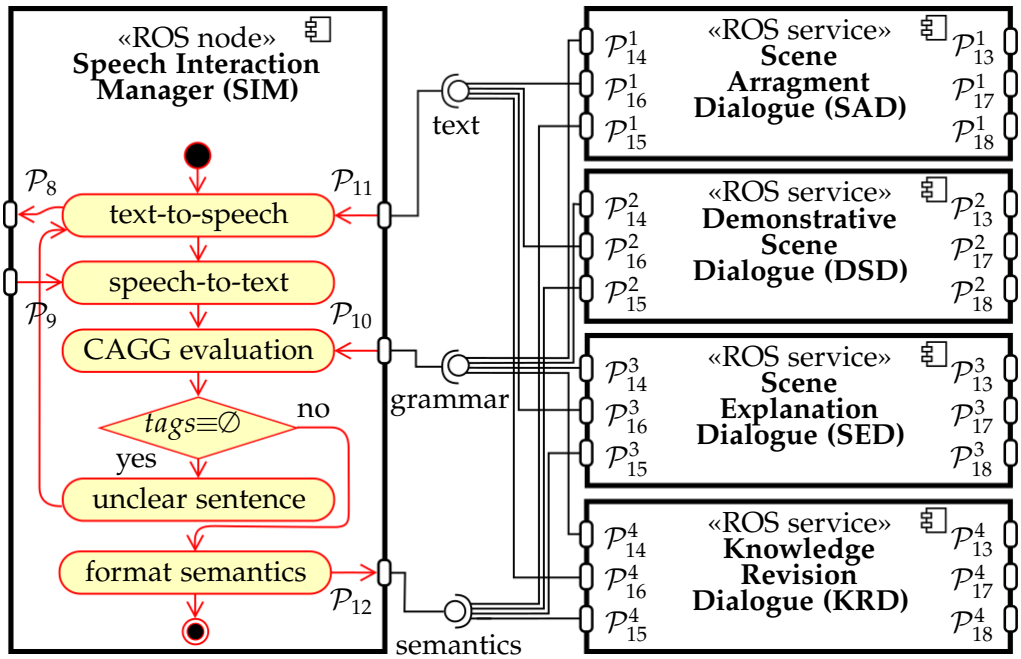


Figure 9.5: The internal components of Dialogue Manager (DM).

audio hardware through appropriate interfaces ($\mathcal{P}_{14}$, $\mathcal{P}_{15}$ and $\mathcal{P}_{16}$). The DM meta-service in Figure 9.2 is based on pairs of such components. The complete architecture can be visualised using both Figure 9.2 and Figure 9.5, where we connect $\mathcal{P}^1_{13}$ to $\mathcal{P}_5$, $\mathcal{P}^2_{13}$ to $\mathcal{P}_{19}$, $\mathcal{P}^3_{13}$ to $\mathcal{P}_{21}$ and $\mathcal{P}^4_{13}$ to $\mathcal{P}_{23}$. Since the purpose of dialogues is to acquire further knowledge to update robot beliefs, we provide each SIM-driver with a connection to AMOR through $\mathcal{P}_{17}$ and $\mathcal{P}_{18}$.

We exploit this design to define four contextualised dialogues implemented as finite state machines, which are discussed below, but the interface can also accommodate more sophisticated dialogue techniques.

9.3.2 Speech Interaction Manager

The Speech Interaction Manager (SIM) is the meta-component managing all low-level aspects of speech-based interaction. In our implementation, SIM is a ROS-based node running with a given frequency, and it is paired with a parallel thread that continuously listens at the microphone ($\mathcal{P}_9$). During each loop, SIM checks $\mathcal{P}_{11}$, and if a *text* is waiting to be processed, it is rendered as an audio signal through the Google Speech API [141], and then played via $\mathcal{P}_8$. If any speech-to-text result is pending, its semantics is evaluated as a sentence via the Concept Action Grammar Generator (CAGG) API [9.5], which exploits Backus-Naur Form (BNF) grammars, *i.e.* collections of rules to assign semantic *tags* to specific words in a text. As an example, a trivial statement can be composed from `[[attributes,...],`

9.5 presented in Chapter 3.

`subject]`, *e.g.* "The green, big, sphere", whereas another rule could be `[attribute,subject,be,object]`, *e.g.* "the blue object is a cone". Therefore, the outcome of the SIM component, through $\mathcal{P}_{12}$, consists of a list of tags representing the semantics of rules that match in a sentence, *e.g.* "blue" is an `attribute`, and "cone" is an `object`. We delegate to SIM-drivers the task of specifying the set of rules to be matched through $\mathcal{P}_{10}$, *i.e.* SIM-drivers are contextualised.

Figure 9.5 shows the SIM node with all ports using asynchronous communication, with the purpose of efficiently using audio channels. Nevertheless, we restrict possible dialogue types assuming that both humans and robots agree on the *don't speak if the other speaks* policy, and that the dialogue does not involve more arguments, *i.e.* there cannot be more than one SIM-drivers activated from $\mathcal{P}_{13}$ at the same time.

9.3.3 Dialogue Types

9.3.3.1 Scene Arrangement Dialogue (SAD)

SKG requests through $\mathcal{P}_5$ the SAD service to wait for the human supervisor to arrange the scene (Algorithm 9.1). Thus, *when* SKG streams a positive Boolean signal to $\mathcal{P}^1_{13}$ (line 9.1.1), the interaction starts. It should be noted that the dialogue driven by SAD does not modify the ontology. It only waits for the human supervisor's trigger, *i.e.* $\mathcal{P}^1_{17}$ and $\mathcal{P}^1_{18}$ are not used. When the scene is ready to be acquired and processed, the human supervisor is supposed to utter a sentence mapped to the `look` semantic tag, and associated with predefined sentences through the rules in grammar $\mathcal{G}_1$ (sent to SIM through $\mathcal{P}_{10}$ at line 9.1.2). Furthermore, at line 9.1.3, SAD requires the robot to notify to the human supervisor that it can process a new scene. Then,

Algorithm 9.1: The Scene Arrangement Dialogue (SAD).

```
9.1.1 when P^1_13 : begin → b ≡ true
9.1.2     P_10(grammar ← G_1)
9.1.3     P_11(text ← "I am looking")
9.1.4     when P_12 : semantic → tag
9.1.5         switch tag do
9.1.6             case G_1.look
9.1.7                 P^1_13(end ←true)
9.1.8             otherwise do
9.1.9                 goto line 9.1.4
```

the system waits for semantic tags provided by SIM through sentence evaluation (line 9.1.4): if the $\mathcal{G}_1$.`look` tag is provided, SAD returns an end signal via $\mathcal{P}^1_{13}$, therefore terminating the service; otherwise, the robot keeps listening to the human supervisor.

9.3.3.2 Demonstrative Scene Dialogue (DSD)

When SIT is not able to classify a scene using the already available scene categories in Φ, the robot builds a new scene category Φ_ϵ starting from the definition of the current scene instance ϵ. SIT delegates to the human supervisor the task of defining the scene category anchor by requiring the DSD service with a signal from $\mathcal{P}_{21}$ to $\mathcal{P}^2_{13}$. Then, DM uses the state machine in Algorithm 9.2 for driving robot behaviour. In particular, DSD requires the robot to ask for a scene name (line 9.2.3), and to evaluate what the human supervisor replies (line 9.2.4). Then, the Algorithm sets the possible SIM semantic evaluation to grammar $\mathcal{G}_2$, defining all the possible *switch*...*case* options based on rules identifying a `name` in a sentence (line 9.2.2). If this occurs, at line 9.2.7 the AMOR service is used to anchor such a word in the ontology to Φ_ϵ. Finally, the Algorithm waits for a consistency check provided by AMOR (line 9.2.8): if it is positive, the dialogue is terminated (line 9.2.9); otherwise, the name is notified to be invalid and DSD is repeated.

9.3.3.3 Scene Explanation Dialogue (SED)

Either SIT has recognised or learned a scene, the robot can describe it and explain how it has been classified. This is accomplished using SED, which requires a human supervisor's feedback about the expla-

Algorithm 9.2: The Demonstrative Scene Dialogue (DSD).

```
9.2.1  when P^2_13 : begin → b ≡ true
9.2.2      P_10(grammar ←G_2)
9.2.3      P_11(text ←"What is this scene?")
9.2.4      when P_12 : semantics → tag
9.2.5          switch tag do
9.2.6              case G_2.name
9.2.7                  P^2_18(manipulations ← [add, Φ_ε, tag])
9.2.8                  when P^2_18 : consistency → c ≡ true
9.2.9                      P^2_13(end ← true)
9.2.10                 else goto line 9.2.12
9.2.11             otherwise do
9.2.12                 P_11(text ← "name not valid")
9.2.13                 goto line 9.2.3
```

nation. As we described in Section 9.2.4, the ontology represents (*i*) the symbolic definition of the most appropriate scene category modelling a scene Φ_*, as well as (*ii*) its implications in terms of other scene categories in the Φ graph. Using two different AMOR queries, SED guides the robot in externalising both sets of beliefs with a similar procedure, shown in Algorithm 9.3 at lines 9.3.2-9.3.6, and at lines 9.3.7-9.3.11, respectively. In the first case, given an actual scene ϵ, the Algorithm iterates over the predicates defining Φ_* (*i.e.* the third column of Table 9.1) formulating a text to be uttered later. In the second case, the Algorithm queries the scene categories in Φ implied by Φ_*, *i.e.* the nodes connected to Φ_* in the scene implication graph, except for the node representing Φ itself (Figure 9.4).

SED is based on a simple grammar $\mathcal{G}_3$, which defines rules for understanding a positive or a negative human supervisor's feedback, as well as specific sentences for requiring the robot to repeat the explanation. First, SED determines the semantic tags provided by SIM evaluations (line 9.3.14). If the human supervisor triggers a `yes`

Algorithm 9.3: The Scene Explanation Dialogue (SED).

```
9.3.1  when P^3_13 : begin → b ≡ true
9.3.2      P^3_17(query ← [classify, Φ, ϵ])
9.3.3      when P^3_17 : response → Φ_*
9.3.4          foreach Predicate pred : Φ_* do
9.3.5              definition ← definition+toSentence(pred)
9.3.6          P_11(text ← definition)
9.3.7      P^3_17(query ← [imply, Φ_*, Φ])
9.3.8      when P^3_17 : response → ancestors
9.3.9          foreach Predicate pred : ancestors do
9.3.10             implication ← implication+toSentence(pred)
9.3.11         P_11(text ← implication)
9.3.12     P_10(grammar ←G_3)
9.3.13     P_11(text ← "am I correct?")
9.3.14     when P_12 : semantics → tag
9.3.15         switch tag do
9.3.16             case G_3.yes or G_3.no
9.3.17                 P^3_18(manipulations ← [add, correctness, tag])
9.3.18                 P^3_13(end ← true)
9.3.19             case G_3.recup
9.3.20                 goto line 9.3.4
9.3.21             otherwise do
9.3.22                 P_11(text ← "I did not understand")
9.3.23                 goto line 9.3.13
```

(or `no`) evaluation, SED requests the AMOR manipulation service to store such information in the ontology (*e.g.* for logging purposes), and terminates the dialogue (line 9.3.18). Otherwise, if the `recap` semantic tag is detected, the robot describes the scene again, while in the other cases humans are asked to repeat their judgement.

9.3.3.4 Knowledge Revision Dialogue (KRD)

As we mentioned in Section 9.2.4, the symbolic representation of scenes created by SKG using RGB-D data must be assessed and if necessary revised by the human supervisor. KRD performs such a revision requiring the human supervisor to listen to robot beliefs and to give feedback for all of them. Referring to the first column of Table 9.1, the robot describes (*i*) object features and (*ii*) spatial relations, which are considered as two consequential phases of the dialogue. Both

Algorithm 9.4: The Knowledge Revision Dialogue (KRD).

```
9.4.1  when P^4_13 : begin → b ≡ true
9.4.2      d_1 ←new X extends Revisioner
9.4.3      P_10 (grammar ← d_1.grammar())      // G_4
9.4.4      d_1.inspect()
9.4.5      d_2 ←new Y extends Revisioner
9.4.6      P_10 (grammar ← d_2.grammar())      // G_5
9.4.7      d_2.inspect()
9.4.8      P^4_13(end ← true)

9.4.9  method Revisioner.inspect()
9.4.10     predicates ← self.queryPredicates()
9.4.11     foreach Predicates pred : predicates do
9.4.12         P_11(text ← self.explain(pred))
9.4.13         when P_12 : semantics → tag
9.4.14             switch tag do
9.4.15                 case self.grammar().yes
9.4.16                     goto line 9.4.11
9.4.17                 case self.grammar().no
9.4.18                     P^4_18(manipulations ← [remove, pred])
9.4.19                     when P^4_18 : consistency → c ≡ true
9.4.20                         P_11(text ← self.help(pred))
9.4.21                 case self.correctingTags()
9.4.22                     P^4_18(manipulations ← [add, self.newPredicate(tag)])
9.4.23                     when P^4_18 : consistency → c ≡ true
9.4.24                         P_11(text ← "I corrected myself")
9.4.25                     else goto line 9.4.12
```

phases exploit polymorphism concepts related to the Object Oriented Programming (OOP) methodology. Algorithm 9.4 shows how KRD instantiates two *objects* $\mathcal{X}$ and $\mathcal{Y}$ extending the same common *class*, *i.e.* `Revisioner` (at line 9.4.2 and line 9.4.5, respectively). Therefore, $\mathcal{X}$ and $\mathcal{Y}$ inherit capabilities from such a class, but they are also required to implement specific *abstract methods*, *i.e.* `queryPredicate`, `explain`, `help`, `correctingTags` and `newPredicate`. Remarkably, since all SIM-drivers use the same software interfaces (Figure 9.5), such OOP inheritance can be shared among all of them. In the first phase, d_1 (an instance of $\mathcal{X}$) uses the `inspect` method to retrieve relevant object features (line 9.4.4). In the second phase, the same method is called on d_2 (an instance of $\mathcal{Y}$) to assess spatial relations (line 9.4.7).

More in detail, for d_1 `queryPredicate` is in charge of fetching the features of all symbols γ_i using AMOR services, as well as the associated anchors to be used in the sentences (line 9.4.10). Then, `inspect` iterates over all query's solutions provided through $\mathcal{P}^4_{17}$, and it describes the representation reported in the first column of Table 9.1 using a procedure very similar to Algorithm 9.4 (line 9.4.12). On the basis of the semantic tags defined in $\mathcal{G}_4$ (line 9.4.3), KRD waits for the human supervisor's feedback (line 9.4.13). If the `yes` tag is detected, the dialogue proceeds with the next belief, since the association between the anchored γ_i and its feature is correct; otherwise, in case the `no` tag is detected, such robot belief is removed from the ontology. Since robot beliefs are associated with specific object features, it is possible to remove them from the knowledge base. This means that if a robot belief about an object's shape were removed, other beliefs would be maintained, *e.g.* in Figure 9.3a, the "cone" would become a "blue object"). When this happens, the `help` method is used to prompt the human supervisor to provide the missing information (line 9.4.20), and therefore SIM generates a signal for $\mathcal{P}_{12}$ (line 9.4.13). If the provided information respects the `correctingTags` case, then `newPredicate` is used to add a corrected belief in the ontology (similar to Algorithm 9.2). If AMOR infers that the human supervisor introduced an inconsistent belief, KRD performs again the revision of the particular predicate. Otherwise, the Algorithm moves to the next predicate to assess.

In the second phase of the knowledge revision process, `queryPredicate` returns the set of spatial relations between pairs (γ_j, γ_k) of anchored symbols (first column of Table 9.1). The procedure `explain` externalises predicates depending on the peculiar relation, for instance `behind(`γ_1`,`γ_2`)`, and then an associated sentence like "The cone has the sphere behind". If the human supervisor considers that such a description is inconsistent (*e.g.* such a relation does not hold in Figure 9.3a), then the procedure `help` instructs the robot to ask for spatial relations among all γ_k and a fixed γ_j. In this case, `correctingTags` are designed according to $\mathcal{G}_5$, which represents `[object,relation]`

types of sentences, *e.g.* "The sphere on the right hand side". When the human supervisor uses such sentences, `newPredicate` identifies γ_k (*e.g.* γ_2 anchored as "the sphere") and the `right` symbols for generating a correct predicate based on γ_j, *e.g.* `right(`γ_1`,`γ_2`)`.

9.4 Use Cases

This Section describes the overall architecture behaviour in three use cases. In our set-up, a Baxter robot is provided with an RGB-D device (connected to $\mathcal{P}_1$), a microphone/speaker (to $\mathcal{P}_8$), and a headset (to $\mathcal{P}_9$) for the human supervisor. We assume that human supervisors are instructed about (*i*) the different dialogue purposes and states, (*ii*) the set of available object shapes and spatial relations, (*iii*) the relevance of object features, and (*iv*) the global reference frame.

9.4.1 Use Case 1

At the beginning, the table and the scene implication graph are empty. Through $\mathcal{P}_5$, SKG activates SAD, so the robot utters "I am looking" (line 9.1.3), and it waits for a reply. In the meantime, PIT is running to track object poses, and estimate their shapes as well as other features, which is used to anchor γ_i symbols. The human supervisor sets the table as in Figure 9.3a and asks something like "What about this scene?" Such audio stream comes from $\mathcal{P}_8$. It is transcribed and evaluated by SIM, which is driven by SAD. SIM identifies the $\mathcal{G}_1$.`look` tag from $\mathcal{P}_{12}$, witch triggers the SAD terminating signal from $\mathcal{P}^1_{13}$ to $\mathcal{P}_5$ (line 9.1.7). At this point, the scene is ready to be acquired, so SKG (through $\mathcal{P}_8$) adds in the ontology the predicates in the first column of Table 9.1, and requests SIT-based reasoning.

As a first step, SIT requires KRD, and Algorithm 9.4 drives the dialogue. The first revision phase starts, and the robot utters "I see a sphere" (line 9.4.12). The human supervisor can confirm this belief saying something like "Yes, that is correct". The dialogue continues with the next belief. The robot says "I see a cylinder", but the supervisor can reply "No, that is wrong", since it is inconsistent (Figure 9.3a). As a consequence, the $\mathcal{G}_3$.`no` case occurs, and the specific predicates are removed by AMOR (line 9.4.18). Then, the robot asks "What is the blue object then?" (line 9.4.20). The supervisor can reply with "A cone", and this generates a suitable semantics for the `correttingTags` of d_1. A corrected predicate for γ_1 is created in the ontology (line 9.4.22). The first KRD phase terminates, since all γ_i symbols are gone through the object shape revision process. The second phase begins with the robot explaining that "The cone has the red sphere on the right hand side" (line 9.4.12). The human supervisor confirms such a belief, using any sentence associated with the $\mathcal{G}_4$.`yes` tag. The dialogue terminates

with KRD returning a signal to SIT (through $\mathcal{P}_{19}$), since there are no more spatial relations in the scene.

SIT represents scene ϵ (Figure 9.3a) according to the description in second column of Table 9.1. It performs DL-based instance checking, but no scene category is available in the ontology, and the implication graph is empty. Therefore, the learning phase starts. DSD takes control of SIM and the robot asks "What is this scene?" (line 9.2.3), and the human supervisor replies with an explicative name, such as "This is Figure 9.3a". Through $\mathcal{P}_{21}$ DSD gives back control to SIT, which learns the Φ_i scene category according with the third column of Table 9.1, and updates the DL-based reasoner for generating a node in the implication graph. SED is then invoked to describe the scene. The robot utters "Figure 9.3a is a scene where at least one cone has a sphere on the right hand side" (line 9.3.6). In this case, no explanation is produced at line 9.3.11, since the implication graph contains one scene category only. Finally, SED requires the human supervisor assessment, which is positive, and SIT ends giving control back to SKG (through $\mathcal{P}_7$).

9.4.2 Use Case 2

The robot waits for the human supervisor to arrange a new scene. The table is set as in Figure 9.3b, then the supervisor can say "Baxter, look!" Similarly to the previous use case, SKG requests SIT, which activates the revision dialogue KDR. The robot states "I see a red sphere" and "I see a green sphere", and then the supervisor confirms twice with sentences associated with the $\mathcal{G}_3$.`yes` tag.

The revision of spatial relations starts with the robot uttering "The green sphere has the red sphere behind it", to which the supervisor can reply "No, this is wrong!", since it is inconsistent with the real situation. Such an incorrect predicate is removed from the ontology, and the `help` method generates the sentence "How are the other objects related to the green sphere then?" (line 9.4.18). The supervisor replies "It has a red sphere on the right hand side". This sentence is evaluated based on `correctingTags`, and `newPredicate` creates the associated belief in the ontology. As a consequence, SIT learns a new scene category Φ_2, which is anchored to the name "Figure 9.3b" through DSD, and it is placed as a node in the scene implication graph. When SIT terminates, the robot can explain that "Figure 9.3b is a scene where at least a sphere has another one on the right hand side". In this case, no scene similarities are discussed by the robot (line 9.3.11), since Φ_2 does not have any connections in the tree except with the initial Φ node.

9.4.3 Use Case 3

In the next scene, the human supervisor introduces a cone as the leftmost object on the table (Figure 9.3c). Using the steps outlined above, the first phase of the revision process makes the robot say "I see a cone", "I see a red sphere", and "I see a green sphere". All these statements are confirmed by the supervisor. In the second phase the robot claims that "The green sphere has a red sphere on the right hand side", and the supervisor agrees (case $\mathcal{G}_3$.yes). The robot keeps saying "The cone has the green sphere on the right hand side, and the red sphere on the left hand side", and the human supervisor replies that "This is inconsistent" (case $\mathcal{G}_4$.no). The robot asks for help saying "How are the other objects related to the cone then?", and the supervisor says something like "It has a green sphere on the right hand side" and "It has the red sphere on the right hand side".

Since the relations between ϵ and Φ_1, or ϵ and Φ_2 are too qualitatively different, the robot does not classify ϵ as an instance of any scene category. Therefore, a new SIT-related learning phase leads to a new scene category Φ_3, and associated with the name "Figure 9.3c" by DSD in the ontology. Afterwards, the robot says "Figure 9.3c is a scene where at least two spheres are on the right hand side of a cone, and at least one sphere is on the right hand side of another sphere". It is noteworthy that the DL-based reasoner infers a scene implication, and the robot states that "Figure 9.3c includes Figure 9.3b and Figure 9.3a" (line 9.3.11).

For the sake of completeness, let us also consider a further arrangement where the human supervisor removes the red sphere from the scene called Figure 9.3c. In this case, given correct knowledge in the ontology, SIT can recognise the scene as an instance of the Φ_1 category. Therefore, the learning phase does not occur, and the description is the same as in Section 9.4.1.

9.5 Conclusions

In human-robot interaction and human-robot cooperation, it is important for humans to have access to the robot knowledge representation system. The paper discusses three use cases in a tabletop scenario where a human and a robot are facing each other. The robot learns, recognises, and describes scenes perceived via RGB-D data, which representation is assessed and if necessary amended by the human through dialogue. We propose and define relevant design/interaction patterns oriented to the use of a knowledge base grounding reasoning techniques, and the adoption of contextualised dialogue management. The discussed architecture allows robots to: (*i*) maintain a representation of salient features of the environment acquired through multiple interaction modalities, and (*ii*) enable humans to instruct, supervise,

and obtain understandable information from a robot at a symbolic level, via speech-based dialogues anchored to the robot belief and reasoning spaces.

10

A Generalised Architecture to Encode, Store, Consolidate, Retrieve, and Forget Structured Experience for Robot's Memories

Abstract

The paper presents a design pattern to build architectures concerning representation of memory items and their long-term management. We present flexible software components for encoding, storing, consolidating, retrieving and forgetting experiences in memory. This paper extends the Scene Identification and Tagging algorithm (SIT) that we designed to perform one-shot structured learning of experiences symbolically represented in an ontology. We design our architecture to support a generic number of instances of the SIT algorithm, which are used for representing the same situation from different perspectives, *i.e.* with different semantics, and we use it to implement a semantic and episodic memory. The architecture was deployed in a tabletop scenario, where an implementation of the consolidating and forgetting functions is presented for episodic and semantic memory. The objective of the paper is to show the support that our architecture can give during the design and deployment of memories for robots.

cognitive memory · robot knowledge ·
consolidate retrieving · forgetting behaviour

Authors

Luca Buoncompagni*, Carlotta Sartore*, and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

Paper to be peer-reviewed.

*These authors contributed equally to this work.

10.1 Introduction

In [168] the memory of an agent is defined as a system able to store information through the interaction with the environment, and it is used to recall experiences in the service of behaviour. In the case of Human-Robot Interaction (HRI), the behaviour of the robot needs to be tuned depending on the application and the design of a suitable memory to support it is critical during development. Typically, there is a trade-off between the complexity of the memory and the richness of its information. On the one hand, using rich memories allows for complex long-term behaviours, but their demand high computation resources and are challenging to design. On the other, simple data structures are more efficient, but they usually support a limited set of behaviours.

In practical scenarios, *e.g.* two persons that cooperatively perform an assembly task, it has been observed that their interaction improves once they become more familiar with each other. In other words, a person also communicates for correcting the projection of the workmate that he or she has in mind. In [81], this phenomenon has been identified as an interaction *gap* that peoples tend to overcome through verbal and not-verbal communication. This process is known to affect their perception of the environment and, their representation of experiences in the memory, which consequently affects their decision processes, *i.e.* the more their beliefs are aligned the better the interaction becomes.

We argue for robots that have memories that can be directly shared with persons, but with a complexity that can be tuned for specific interaction patterns as well. Therefore, a robot should have a memory that allows to process data and represents the environment in a way that is familiar to the users. In this paper we want to develop a robot that communicates with the user in order to align its representation of the environment, and that can maintain such representation for being recalled in a long-term manner. After some experience, the robot might remember several similar situations from memory, in this case, it should be able to rank them in order to identify a relevant set of information that is required for a particular context. This is not only a type of representation that is familiar to a user, but it also reduces the complexity of the system, and ambiguous behaviours, because not relevant information can be disregarded based on contextualisation [30].

In human-computer interaction, the problem of tuning the information needed for maintaining the symbols through which the user interacts is typically addressed with the iterative development process [82]. In particular, such a process is made of three phases that are iteratively performed until the end of the development. The fist consist in developing a part of the system, in the second experiments are performed, and in the third feedbacks are evaluated form planning the first phase of the next iteration. Remarkably, during the latter

phase, small system's changes should be scheduled in order to make the user's feedbacks guiding the development as much as possible, *i.e.* the more iterations are done during the development process, the better the interaction is likely to be.

We aim to use the iterative development process for guiding the design of the representation that a robot should have for performing specific tasks jointly with a person. For achieving such a development, we require to perform as many changes to the system as possible, with the purposes of taking comparable measurements throughout iterations. Such a complex development process should be supported from a *modular* system, that allows for extensions and rearrangement. In this paper, we present a modular set of components that can be used for supporting the implementation of different memory management policies. We present the implementation of the first developing iteration deployed in a tabletop scenario, where we focus on the design of an architecture that would support all the further iterations for such a scenario.

In the next section, we give a more detailed background for our work, from both technical and cognitive perspective, as well as our contribution. In Section 10.4 we formalise the requirements and the operations that the robot should perform with a contextualised memory, and in Section 10.5, we give an overview of the tools we used and of the architecture. In Section 10.6 we present a general-purpose knowledge representation, which is used for implementing the functionalities of a generic memory type in Section 10.7. We specialise the latter formalisation to implement a semantic and episodic memory through two ontologies in sections 10.8 and 10.9. Section 10.10, presents our architecture, its modules and general purposes interfaces that we obtained with our design pattern, which is discussed in details through examples for a tabletop scenario.

10.2 Cognitive Memories and Related Work

In cognitive science, the memory is a structure made by contextualized sub-systems containing past experiences and subjected to discrete functions such as, *encoding*, *storing*, and *retrieving* [168]. In particular, each agent perceives the environment with different prospects [161], which are encoded in *items* (or chunks) of knowledge, and stored in a dedicated part of the memory. The retrieving function accesses the memory to recall, also through internal simulation [161], the items stored with different contextualization that are relevant for the current situation. The perceived information and the retrieved items are combined in another dedicated part of the memory, which is used for taking decisions efficiently, and it is maintained through item refreshing (*i.e.* *rehearsal*) for the time in which a particular situation deserves the agent's attention.

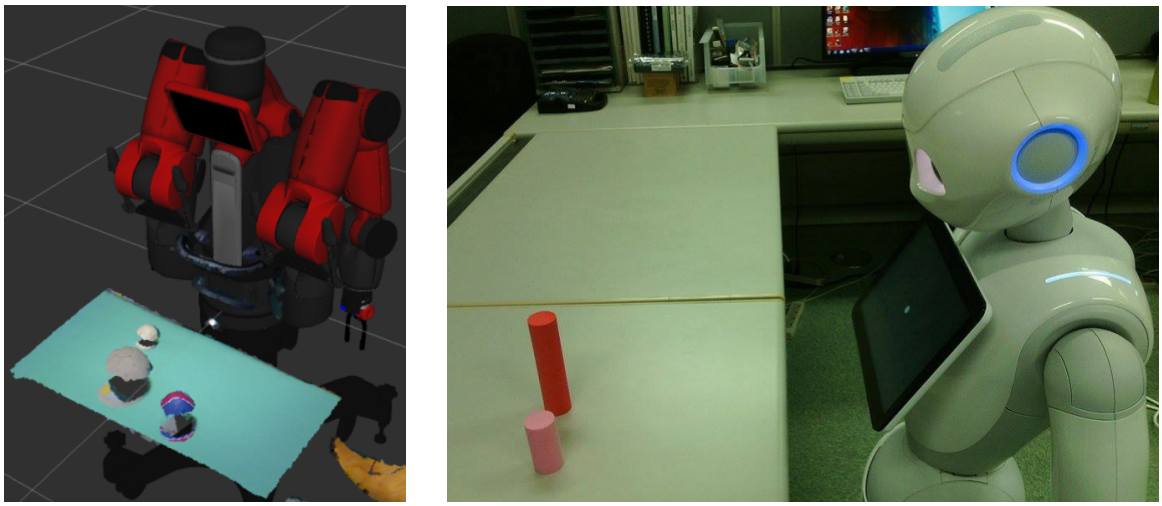


Figure 10.1: The tabletop scenario in which we tested our architecture.

In particular, in [153] the *maintenance* rehearsal is identified as frequent items processing demanding low attention, and this process is useful for immediate retrieving, but it lacks for long-term representations [68]. On the other hand, the *elaborative* rehearsal processes the knowledge deeper than the maintenance rehearsal since it implies association among items and reasoning about their meaning [67]; this process helps in mapping long-term knowledge. In [66], *knowing* is associated with the maintenance rehearsal, while *remembering* to the elaborative rehearsal. Those concepts are related to learning and classifying abilities, which are performed both in a *one-shot* manner (*i.e.* only one experience is enough to classify similar situations in the future), and through *consolidation* (*i.e.* repetition of items regarding some experience). In [51] consolidation is described as a process that stabilises an item after a preliminary acquisition; for instance in [54] it has been observed that while sleeping memory consolidation is undertaken. Also, during the consolidation of items in memory, emotions affect the way in which they are stored [107], and a reconsolidation [104] processes updates items in memory, which occur also during a retrieving task.

Nowadays, robots can learn from repetitions in a data-driven manner for recognising peculiar characteristics of the environment, and for performing actions, *e.g.* for detecting objects and identify a grasping pose from images through deep learning [94], or for encoding, storing, retrieving and predicting episodes [131]. Also, there is evidence [86] where reinforcement learning has been used for representing knowledge in complex spaces through the rewards toward desired behaviours, which is based on internal and external stimuli, *e.g.* in order to make a robot riding a bike [127]. Those learning approaches are computationally long but effective with row data, and they well generalise the functions for managing the memory in the service of

specific behaviours. However, if a robot uses the models generated with data-driven approaches, it is typically not able to explain to the user its beliefs and decisions. Therefore, those approaches well perform for low-level behaviours, that are so complex and embodied to not being communicable, *e.g.* nobody can learn how to ride a bike from a book.

For high-level behaviours (*e.g.* planning, or instructing), robots can also rely on data structures used by modules that reasons on events and take decisions in a probabilistic manner. The model they produce is expressed in a form that a user might understand, and their design can be driven by data as well. In particular, the Hidden Markov Model has been used for design memory in order to perform prediction based on experiences to support planning [99, 142].

On the other hand, purely symbolic representations can be designed in order to exploit logic reasoning. The Description Logic (DL) [11] formalism is becoming a standard for reasoning, but also other memory structures have been deployed in databases, *e.g.* [115, 124, 150], exploiting also sub-symbolic formalisms [14]. The Ontology Web Language (OWL) [138] gives a standard logic representation for developing general purposes DL reasoning algorithms. DL has been designed for contextualising knowledge based on its semantic, and the reasoners exploit this formalism to retrieve information through logic queries, which can be expressed in sentences, *e.g.* as shown in [91]. Under some expressibility constraints, it could be possible to ensure scalability of the retrieving query for long-term applications, but generally, symbolic reasoning is a complex task [49]. Therefore, for efficiency purposes, we would argue for reasoning processes that do not evaluate a unique and cumulative memory.

It has been observed that humans contextualise their knowledge and focus their attention in order to retrieve information that is relevant to a situation. This behaviour decreases their cognitive *load* since not relevant information is disregarded, as observed in [16], where the load is defined in terms of time and retrieving demands. In particular, state of the art architectures divides the knowledge in the Short Term and Long Term Memories (STM and LTM respectively) [112]. The former is limited but faster, while the later is bigger but slower, not only in the retrieving process, but also the encoding and storing functionalities are more complex, and they might also involve experience consolidation. The retrieving and storing functionalities in the STM is faster than in the LTM, but the items contained in the STM decay over time quickly if a rehearsal process is not undertaken. As the taxonomy in Figure 10.2 shows, the STM has been further divided in the Working Memory (WM) in [109], and Sensory Memory in [8]. The Sensory Memory are buffers that contain perceived data streams encompassing the iconic, echoic, and haptic information; *i.e.* it receives row external stimuli. In [71] also a part of the STM that is uniquely related to motion have

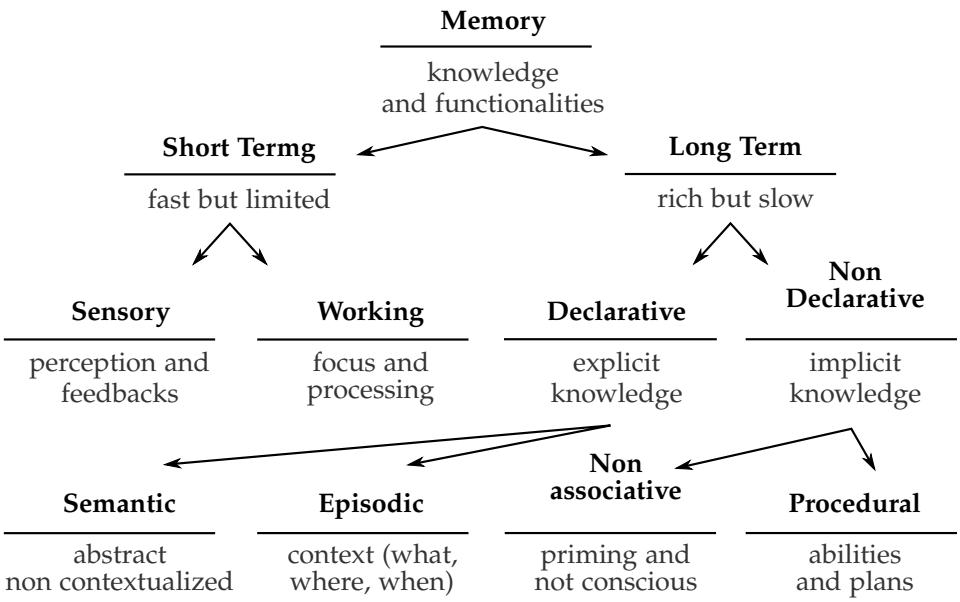


Figure 10.2: A simplified taxonomy of memory types [97, 148].

been identified as the motor memory, while in [13] the episodic-buffer has been identified for containing the integration of different sources in a limited time window. Also, WM contains some knowledge encoded in the Sensory Memory through attentive filtering [72], as well as items retrieved from the LTM, and both are used for reacting to the current situation. In the WM, knowledge is used to perform internal simulations for predicting the overcome of actions, and activate goals. The retrieved items from the LTM that are evolved in the WM at a particular time get consolidated, since emotions and perception of the current situation affect the representation of items in the LTM.

Moreover, in [39] another important distinction was made among the declarative (*i.e.* explicit) and not declarative memories (*i.e.* procedural or implicit). The first encompasses explainable experiences, which are usually learned in a one-shot manner, while the last cover procedural abilities (*e.g.* bike riding), as well as not associative learning (*e.g.* priming behaviour and not conscious memory). In particular, declarative type of memory encompasses the Episodic Memory (EM) and the Semantic Memory (SM) [159]. In EM episodes are represented through a time (*when*), a representation of the environment (*where*), and information about its content (*what*), that characterise a situation with respect to the context in which it has been experienced. On the other hand, the semantic memory contextualises the knowledge from a different perspective, which is more abstract and does not depend on the current situation.

In architecture for robots, implementations that emulate the STM and LTM have been largely used for in order to process data streaming, usually in fixed time window or memory size. In particular for STM, due to time decay of items, the efficiency of encoding storing

and retrieving functionalities is crucial, and many systems optimise the design of such a memory structure based on their application and setup. Declarative memories have been deployed in robotic applications as well. For instance, [115] proposes a system based on a database to generate knowledge in the SM through a perception memory and learning behaviours. On the other hand, [114] identifies the requirements of an episodic-like memory for cognitive agents. Such a representation should be structured in a unique space that allows reasoning on nested and overlapping episodes, that are learned in a one-shot manner, and maintained for long-term. Some robots exploit both semantic and episodic memory since using both the representation improves the experience retrieving [75]. For instance, in [119] an episodic item in EM is extended with a generalisation of spatial and temporal relations in order to represent the same experience also form a semantic perspective in SM.

Among the operation of encoding, storing, retrieving and consolidating operations that the memory performs on its knowledge, also forgetting is an important behaviour, which is considered to be a necessary cognitive process for accomplishing complex and long-term behaviours, as also noted in [15]. In particular, temporarily or permanently forgetting, allows the robot to disregard not relevant experiences, which might help it as far as computation complexity for taking decisions are concerned since ambiguous information from memory would not be retrieved. Furthermore, if the robot retrieves only some experiences from a particular context and, if the retrieved items are aligned with the beliefs of the user, this might also improve the interaction. However, select the experiences to recall is not a trivial task, and we believe that it is an issue worthy of attention from the robotics community. This issue could be addressed through contextualisation techniques, *e.g.* probabilistic or logic approaches to recognise activities given distributed knowledge, as presented in [83] for instance. However, in the literature, those approaches usually do not involve experiences and are based on a static set of situations that are modelled since relevant to an application. On the other hand, cognitive theory addressing the problem of forgetting has been proposed, and they could be inspiring for building suitable forgetting behaviour for robots. However, in the literature there is not a comparison among different approaches for HRI scenario; thus, there is no clear evidence of the benefits or limitations they might have.

Humans forget or modify memory items spontaneously which is affected by encoding, storing, and retrieving tasks occurring in the memory [102]. In the STM items are supposed to last for a small time window if a rehearsal operation is not performed; otherwise, they are forgotten. Through attention, the rehearsal process can be such to *move* items from the STM buffer into the LTM, where the knowledge it represents is consolidated. However, persons experience to forget

also items that are considered to be stored in LTM, but in this case, the process seems to be more complex than a simple buffer of items as in the STM. Also, persons experience to retrieve an item if they spent effort in remembering a situation that initially seems to be forgotten, which highlights the dependence among the different memory functionalities, as well as their complexity. Interesting enough, there are techniques to measure human forgetting behaviours. For instance, [36] proposes the prompted recall based on free remembering tasks concerning items of a list shown through stimulus. It also presents the relearning method, which measures the amount of training to recall the items of a list, as well as a measure based on the number of items that could be recognised through their re-observation.

Apart from organic causes, such as dementia, three main theories tackle the issues of defining a forgetting operator that models humans behaviours. The context-dependent memory approaches have been proposed based on the idea that encoded knowledge about an item to be retrieved can be corrupted, and consequently, such an item cannot be retrieved, *i.e.* it seems to be forgotten. This theory involves the STM and LTM [145] and suggests that forgetting increase with age since our ability to encode information decreases. Also, it involves the context as a component that affects our encoding ability, including emotional states, which might make the knowledge in memory not well represented for retrieving an item.

Moreover, the interference theory suggests that item from the LTM cannot be recalled in the STM to be retrieved since there could be inconsistencies among the different representations. The theory dived in retroactive and proactive [160], as well as output interference. Retroactive interference occurs between newest and old items, and an example could be the interference between the names of a new partner against the name of an ex-partner. While with proactive the interference between old and new items is considered, for instance, when an ability as not performed for a long time it is difficult to recall it. Instead, output interface considers the interface that the action to retrieve an item has to the representation of such an item [158], an example could be when the action to remember a list of items allows to recall some of them but inhibits the retrieving of the others.

Last but not least, in the trace decay theory [80] items are supposed to have a trace that degenerates over time and, due to some activation levels, the trace could be forgotten. This theory applies to STM and LTM, but concerns have been raised about the long-term representations since the knowledge to perform some abilities (*e.g.* ride a bike) do no seems to decay over time. This suggests that for LTM memory other factors should be considered in order to model a decaying trace. Remarkably, some of those theories have been used for implementing robot with forgetting behaviours. For example, the ACT-R cognitive architecture implements a trace-based decay approach [6], while in [27]

the interference among items is computed based on their similarity evaluated as the distance among point in a particular space. The union of those two approaches is presented in [63].

In the literature, there is a broad agreement on the different purposes of memory: it is a representation composed by different contextualization of knowledge, that are interwoven among each other, and supports functionalities. In particular, memory items encoded from different perspectives are stored affecting each other, and the different retrieved experiences are merged in the service of behaviours. This implies that each memory type represents structured knowledge subjected to specialised functions which are used in synergy by all the components implementing some behaviours of the robot. In this way the same experience is represented differently and, in order to maintain them over new stimuli, the memory should be able to associate the current situation with a previous experience available in different parts of the memory.

For managing a memory, *i.e.* orchestrating the computation of all its functionalities, it is important to take into account that knowledge is subjected to different contextualisation. This implies the robot to be able to associate items with multiple representations for generating a specific behaviour, which might involve further memory manipulations. Several cognitive architectures have been presented and improved for generating and using knowledge structured through the implementation of some cognitive concepts introduced above. Unfortunately, in [170], where an extensive comparison of cognitive architecture is presented, there is an evident lack of integration and techniques validation. Nevertheless, for efficient integration, we argue for a systematic comparison of the techniques first, and later we would synergistically integrate them in order to exploit their pros and compensate their drawbacks. With this idea, we want to address the issues of comparing memory representations and different operations for managing their items, and we preliminary aim to design an architecture to support such comparisons. This paper presents the architecture we design with that purpose, and we discuss its flexibility as far as memory item management is concerned.

10.3 Rationale

Our architecture is based on the Scene Identification and Tagging (SIT) algorithm, that we present in detail in Chapter 7. The algorithm has been designed to perform one-shot structured learning of scenes shown to the robot over time. In particular, the algorithm uses symbolic formalism to return a graph of experiences given some input facts. On the one hand, the inputs are given in accordance with a semantic interface. On the other, the output graph contains nodes that are models of past experiences matching the given facts, while their edges

represent the relations between different experiences. In Chapter 7, we based our algorithm on an OWL ontology, and we obtained output graphs where edges were representing experiences similarities. Thank OWL reasoning, we obtained an acyclic oriented graph of experience, that has a root and some leaves, where deeper the node more complex the experience it models would be.

In Chapter 7, we showed the behaviour of the algorithm based on a simple tabletop scenario where objects were spatially arranged, and we discussed the generality, limitations and complexity of our implementation also considering other scenarios. In this paper, with showing purposes, we use a simplified scenario based on the one presented during the formalisation of the SIT algorithm. Nevertheless, we aim to discuss and motivate the deployment of our architecture for more general scenarios. A limitation we highlight in Chapter 7 is that the algorithm is not robust to perceiving errors, and we assumed the input facts to be always consistent with the environment. To address this issue in 8 we presented an extension of the algorithm based on a fuzzy ontology, which support reasoning under uncertainties. We shown that SIT is equivalent to its fuzzy formalisation when all the membership values are either 0 or 1. For the sake of clarity, in this paper, we consider the crisp implementation of the algorithm, but our approach can be applied to the fuzzy implementation of the algorithm as well.

Furthermore, the algorithm is provided with a novel and expressive semantic interface. Differently, from the standard interfaces between components of software architectures, which are in charge to specify the semantic and the data that two components can exchange consistently. The semantic interface assumes a centralised knowledge representation, *i.e.* the robot's memory, that is shared among many components of the architecture, which exchange mainly triggering signals. When a component is triggered, it searches in the memory for knowledge to process without knowing the provenience of such information.

For doing so, the architecture should assure that the component can find processable information in the knowledge representation. Thus, we consider that each component accessing the knowledge requires a semantic interface, which describes the semantics of the information that has to be in the memory to process such a component consistently. Therefore, two components exchange information through the knowledge representation. This implies that similarly to the inputs also the outputs of a component are typically knowledge structures in the memory, which are supposed to respect the semantic interface of other triggered components, which requires such a structures.

10.3.1 Contribution

The paper presents a design pattern to build software architectures for maintaining a structured and contextualised memory. We based our method on an extension of the SIT algorithm through which we perform the encoding, storing, retrieving, consolidating, and forgetting operation on parts of the memory.

In particular, we consider the memory as divided into *types*, *e.g.* as shown in Figure 10.2. We want assign an instance of the SIT algorithm for implementing each memory type. Each SIT instance is provided with a specific semantic interface, which is used to structure experience based on some aspects of the environment. Since SIT structures experience based on OWL reasoning, we consider the memory as a collection of ontologies, one for each SIT instance, that represent the same experiences but with different purposes.

Through SIT instances we generalise the implementation of a memory type, and we give guidelines for defining the common operations that should be used for managing its knowledge over time. In particular, we consider memory types to distinct structures of experience, which are *associated* among each other. The knowledge in each memory type is maintained through the *encoding*, *storing*, and *retrieving* functions, which are based on extensions of the perceiving, classifying, and learning functions defined in the SIT algorithm. Also, we define the *consolidating* and *forgetting* functions, which have not been considered in the original formalisation of SIT. For demonstration purposes, we specify our general pattern for maintaining a semantic and an episodic memory through some software components. The paper highlights the extensions on the general pattern we made for maintaining, in different memory types, the knowledge about some situations, *e.g.* scenes in a tabletop scenario.

Remarkably, this paper does not focus on the design of effective memory functionalities, *e.g.* for consolidating and forgetting experiences. Instead, it focuses on an architecture that supports their implementation. Indeed, since the design of those functions is far from trivial, we aim to build an architecture where different approaches can be implemented with evaluating purposes. Nevertheless, we discuss an implementation of the architecture for our preliminary scenario evaluated in a real setup.

Therefore our objective is two-fold, we want to (*i*) design software components of an architecture that maintain structured experiences, and (*ii*) support the investigation of consolidating and forgetting techniques through a significant set of information for implementing them. In particular, our architecture contains placeholders, *i.e.* interfaces, designed to be flexible enough for testing different approaches for representing and managing experiences in a robot's memory. Since we want to implement robot's that interacts with users, the possibility to

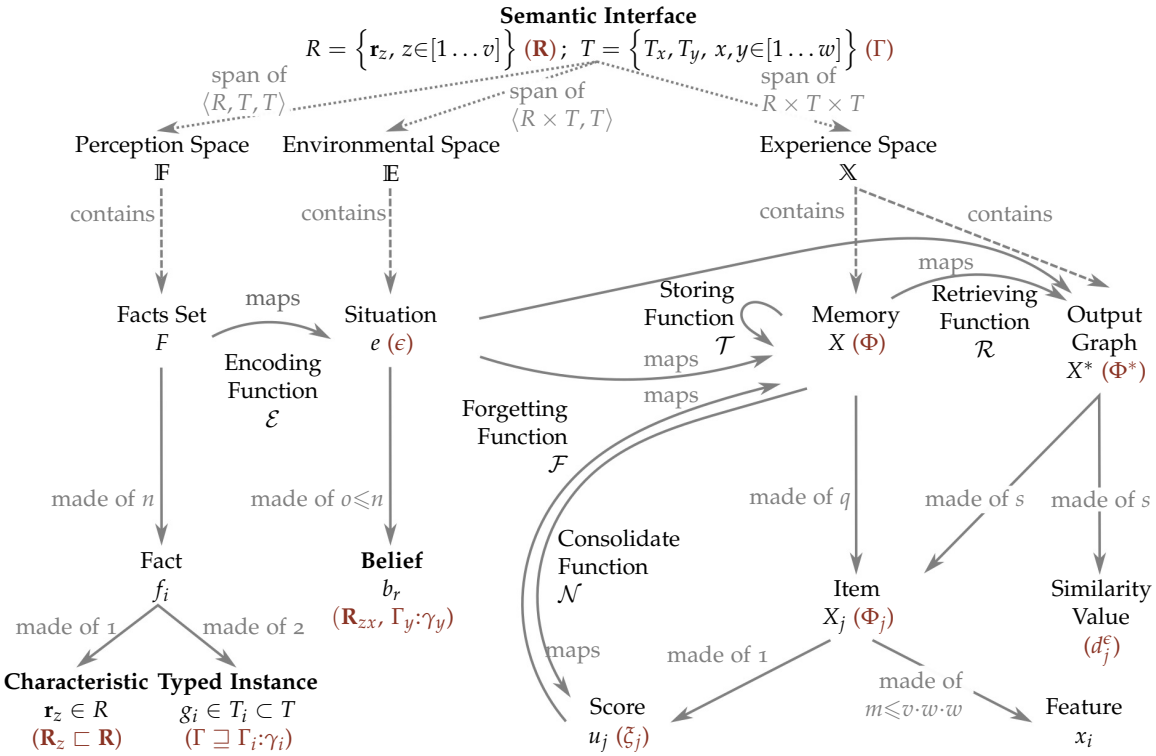


Figure 10.3: A glossary of the symbols (in black), and their relations (in grey), defined in this paper for representing a type of memory based on the SIT algorithm. For a quick reference, we show in red some elements of the ontology that are relevant for implementing the related black symbols, which are motivated during the description of the architecture.

test different techniques in a real scenario allows also evaluating his or her behaviour, which is desirable for developing robot's knowledge toward a real application.

10.4 Problem Statement

Based on the SIT algorithm, we want to define the knowedlge and the computation required to implement the *encoding* $\mathcal{E}$, *storing* $\mathcal{T}$, *retrieving* $\mathcal{R}$, *consolidating* $\mathcal{N}$ and *forgetting* $\mathcal{F}$ operations. Those functions are supposed to maintain a graph of memory *items*, that we call *memory* ${}_pX$. The graph contains nodes that are items ${}_pX_j$, whose model experiences with a semantic that is specified for one part of the memory, *i.e.* the p-th memory type, *e.g.* the short-term, semantic or episodic memory.

More formally, we consider the whole memory as a set of graphs $M = \{{}_pX,\ p \in [1 \dots h]\}$, where $\bigcap_p^h {}_pX = \emptyset$. Nevertheless, we consider that some nodes in the different graphs might be related among each other through an *associativity* operator, which relates items modelling the same experience among the h graphs, *i.e.* maintained with different semantics. To simplify the notation we do not indicate the p-th index when we consider symbols related to the same, but generic, memory

type. Otherwise, when we compare symbols from different parts of the memory, we explicitly indicate it with the left subscript.

The structure of the symbols that we want to represent in M over long-term applications is shown in Figure 10.3. In particular, given input *facts* we represent h *situations* e encoded with different semantic. Then, the storing function of each p-th memory should be used for structuring models of the situation based on previous experiences. This function requires learning, that we identify as the operation of generating a model from the beliefs of a situation, *i.e.* a memory item X_j. Given a structure of items X, the retrieving function is used for recalling some models of a given situation e, *e.g.* encoded through dialogues.

While those functions are related to external stimuli, *i.e.* data acquired from sensors, the consolidating and forgetting functions are computed internally, *i.e.* based on an internal representation of an item's *score*. We consider the score to represent the importance of an item with respect to the other in the memory, and we design the consolidating function to compute the score of each item X_j. While, the forgetting function uses it to classify the items, which are eventually removed from the memory. We preliminary consider three possible levels of score that classifies the item either in the *high*, *low* or *weak* level. An item with a high score is supposed to be directly retrievable given e, while an item with a low score can be retrieved only when future experiences are such to classify it as having a high level, *e.g.* after repeated retrieving operations. Only items with weak scores are removed from the experience graph of the robot.

On the other hand, the consolidating function is in charge to compute scores that rank items X_j, and this effects the retrieving and forgetting operations. We suppose the score to be dependent on an expressive set of parameters that can be used for implementing different memory management policies. Preliminary, we consider the score (*i*) to increase in case of recurrent stored, or frequently retrieved, situations, and (*ii*) based on the internal state of the memory, *i.e.* on the relations among items and their definition.

It is important to notice that the concepts presented in this paper, and summarised in Figure 10.3, are similar to the one used to formalise the SIT algorithm in Chapter 7, but they refer to a different semantic. For clarity purposes, we maintain the same symbols involved in the SIT formalisation in this paper, but we remap their names since they have different semantics, even if they share the same representation in the knowledge base. The next section introduces the functionalities of SIT adapted for this paper, while Section 10.4.2 defines the functions that this paper presents based on the former.

10.4.1 SIT Functionalities

Through the comparison Figure of 7.2 with Figure 10.3, it is possible to observe that the perception space $\mathbb{F}$ considered in this paper is equivalent to the one designed during the formalisation of SIT. [...][10.1]. In this paper, we implement the retrieving and storing functions on a combination of $\mathcal{C}$, $\mathcal{L}$ and $\mathcal{S}$. This because, based on classification, we want to assess the state of the memory while with the learning and the structuring function we want to consider new items. Moreover, we want to add in the SIT representation of each j-th item a score value u_j, which should be computed from the consolidating function $\mathcal{N}$ and used during the retrieving $\mathcal{R}$ and forgetting $\mathcal{F}$ operations.

10.1 in the original version of the paper we recall the classifying $\mathcal{C}$, learning $\mathcal{L}$ and structuring $\mathcal{S}$ functions of the SIT algorithm. In this paper we define the encoding *function $\mathcal{E}$ as the perceiving function $\mathcal{P}$ defined in Equation 7.7. We also recall the concept of the scene ϵ and its beliefs, as well as a category Φ_j and its features. Finally, it introduces the concept of experience graph Φ and output graph Φ^*. All the details are presented in Chapter 7.*

10.4.2 SIT extensions

We define the *consolidating* function $\mathcal{N}$ to compute the score of each j-th items u_j, *i.e.* a real number in $[0, 1]$, based on the memory X and such an item X_j

$$\begin{aligned} \mathcal{N} : \mathbb{X}, \mathbb{X} &\to [0, 1] \\ X, X_j &\mapsto u_j. \end{aligned} \tag{10.1}$$

Remarkably, with this definition we supposed in the memory X all the knowledge needed to implement the consolidating function.

The consolidating function uses a specific semantic interface to represent the scores and triggers the *forgetting* function $\mathcal{F}$

$$\begin{aligned} \mathcal{F} : \mathbb{X}, [0, 1] &\to \mathbb{X} \\ X, d_j &\mapsto X, \end{aligned} \tag{10.2}$$

which updates the graph X given the score of the j-th items, *i.e.* it might remove X_j from the memory X.

Furthermore, we consider the *storing* function $\mathcal{T}$ as a map between beliefs about a situation e and the features of an item X_j. If encoded facts can be classified through $\mathcal{C}$, *i.e.* the output graph $\bar{X} \neq \emptyset$, than the situation matches the features of some memory items. Otherwise, the memory does not contain any experiences about such a situation, that should be learned. In the latter case, we apply the learning $\mathcal{L}$ and structuring $\mathcal{S}$ functions to generate a new node in the memory X, which is the item modelling the situation e.

When an item is learned from the beliefs about a situation, we consider the storing function to initialise its score. While, if storing function classifies e in some items previously stored $\bar{X}$, than we suppose $\mathcal{T}$ to represent this event since we consider the scores to be affected from frequently experienced situation. The consolidating function uses this knowledge represented in the memory, and it is triggered by $\mathcal{T}$

and $\mathcal{R}$. The retrieving function $\mathcal{R}$ has a behaviour similar to $\mathcal{T}$ since we assume the score being affected by situations that are frequently demanded.

In particular, $\mathcal{R}$ is a function that returns a structure of items X^* given the memory X, and a situation e

$$\begin{aligned} \mathcal{R} : \mathbb{X}, \mathbb{E} &\rightarrow \mathbb{X} \\ X, e &\mapsto X^*. \end{aligned} \tag{10.3}$$

Its definition is equivalent to $\mathcal{C}$ in Equation 10.3, but the behaviour is different. Differently from $\mathcal{C}$, where we assume that the facts exhaustively represent the environment, $\mathcal{R}$ might be computed for a subset of facts. For instance, consider the case where a situation should be retrieved given only a partial description of the facts that were involved.

We supposed each memory type in M to be supported with a dedicated instance of the SIT algorithm; therefore, we can identify h graphs ${}_pX$, which are managed through specific implementations of all the introduced functions. Given some facts, we represent h situations ${}_pe$, and if those are used with retrieving purposes, we obtain h graphs X^*. Each node in the following graphs have a score and a similarity value, and some nodes can be associated among graphs. Within the same graph, scores rank the item among each other, while the similarity value indicates how well each item describes the given situation.

10.5 System Overview

This paper presents an architecture[10.2] where all the functions introduced in Section 10.4 can be computed, based on some knowledge. In particular, functions are implemented from different components of the architecture that we generalise for a p-th memory type. To motivate our design pattern, this paper presents a simplified scenario where semantic and episodic structures of experience are maintained over time based on the same implementation. For each of those two memory types, we show how to exploit our design pattern to implement a large set of different implementation of the storing, retrieving, consolidating and forgetting functions.

10.2 *a preliminary implementation is available at* `https://github.com/EmaroLab/ETNRF-SIT-memory`

In particular, we deploy our architecture in a tabletop scenario involving a robot and a user. The robot is a passive observer that structures its memory based on situations that are generated by the user, which arranges objects on the table. Therefore, the robot experiences spatial scenes as shown in Figure 10.1 for instance, which are retrieved by the user through dialogue in a simplified English.

In our scenario, the robot should rely on three modalities. One is based on an RGB-D camera that is used to acquire objects on the

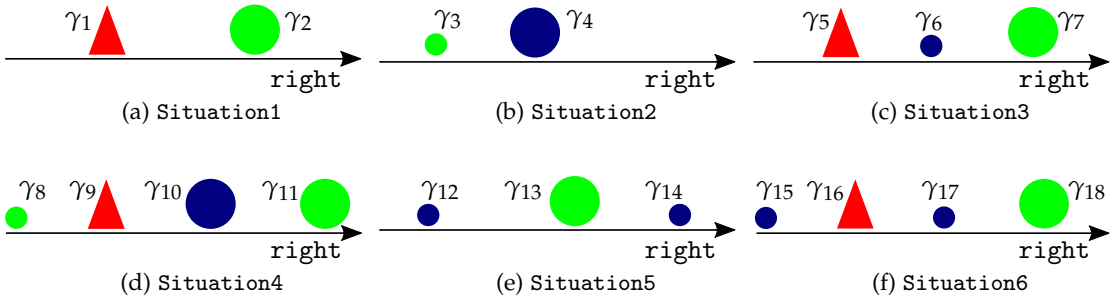


Figure 10.4: Examples of situations that the robot experienced.

table. While the other is based on a microphone, which is used for acquiring some statements about the scene to be retrieved. The third modality is based on speakers and a monitor, which the robot uses to communicate with the user.

Figure 10.6 shows the software architecture that implements the system we obtained through the proposed design pattern deployed in our scenario. The architecture encompass:

- a *knowledge representation* containing the one ontology for each memory type,
- a *perceiving module* providing encoded situations,
- *external reasoners* implementing the functions based on external stimuli (*i.e.* storing and retrieving),
- *internal reasoners*, implementing the function based on internal stimuli (*i.e.* consolidating and reinforcing), and
- a *hybrid* perceiving-acting module, which manages the dialogues with retrieving purposes.

As mentioned our architecture is based on multiple instances of the SIT algorithm, which provides a semantic interface that defines the structure managed for each memory types, *i.e.* the semantic of each graph ${}_{p}X$, which is maintained in a dedicated ontology. This paper shows how different SIT interfaces can be used for maintaining specific graphs based on the same situations, *i.e.* ${}_{S}X$ and ${}_{E}X$: the *semantic* and *episodic* memory respectively. Although the design of expressive semantic interfaces for a particular scenario is out of the scope of this paper, we discuss a possible representation for semantic and episodic items as far as simple spatial relations are concerned. However, the paper mainly focuses on the components to be used in order to implement the functions introduced in Section 10.4.

10.6 Memory Representation

Our architecture is based on h instances of SIT, whose have functions specialised to maintain specific memory types ${}_pX$, whose are represented in dedicated ontologies. In each ontology, we expect SIT to maintain knowledge about the experience with different representation. Therefore we configure each SIT instance with a specific semantic interface, which defines prior knowledge that is used to represent facts. This section recalls the properties of a semantic interface which specifies the knowledge that SIT requires in the ontology, and presents the structures that are in common with all the ontologies of our architecture.

Initially, we introduce the DL formalism that we require to detail the implementation of the architecture. Then, we introduce the knowledge representation involved in the SIT algorithm in Section 10.6, and two upper-ontologies defining knowledge included in each ontology representing a memory type ${}_pX$. The first, in Section 10.6.2, represents the score *level* of each item in the memory, which is computed by the forgetting function. The second upper-ontology, in Section 10.6.1, defines the representation of the item's *state*, which are provided from the storing and retrieving functions, and are requested for consolidating the items. Those representations are used to assure in the ontology that the knowledge required by our architecture is available after the process performed by some other components.

Description Logic Primer

[...][10.3]

10.3 an introduction to DL can be found in Chapter 5.

SIT Ontology Interface

An instance of the SIT algorithm requires to know the symbols that could be available at runtime in the ontology, *i.e.* a IRI identifier or *name* $\mathcal{N}$. Those symbols should have a specific format in order for SIT to perform consistently among its phases, which involve the perceiving, learning, structuring and classifying functions. [...][10.4]

During learning, the number k_{zxy} is obtained by counting the beliefs of the situation ϵ from which we want to generate an item Φ_ϵ. After this process SIT requires the structuring function, which relies on OWL reasoning to check the consistency among the item's restriction previously stored. The result of the reasoning task is an acyclic oriented graph that we identify as a memory X, where each node is a class in the ontology, *i.e.* a memory item. The graph always involves a node Φ, which is the root and represents an empty situation, as well as some leaves. The nodes in the ontology are structured based on the logic inclusion operation ($\sqsubseteq$) and are such that, given a generic node, its

*10.4 here we describe the semantic interface defined in Chapter 7. We introduce that SIT learns scene categories (*i.e. *items) through reification based on the names ($\mathcal{N}$) of facts contained in an ontology. The latter are defined as relations $\mathbf{R}_z$ characterising two instances γ_x and γ_y of a non-ambiguous type Γ_x and Γ_y.*

parents model simpler situations, while the children represent more complex models.

As a measure of complexity among items, in Chapter 7 we defined the item cardinality as

$$^{j}d = \sum_{zx,y:\Phi_j}^{m} k_{zxy}, \tag{10.4}$$

which is the sum of all the restrictions for all the features. Also, given some facts we define the situation cardinality as the number of beliefs that the encode function generates n. In Section 7.5.5 we discuss the similarity value, which is define as the ratio of the item cardinality and the situation cardinality of ϵ, *i.e.* $^{j}d/n = {}^{j}d_{\epsilon}$.

10.6.1 Upper Ontology for Item's States

In each memory type, we use SIT to maintains a graph of memory item ${}_{p}X$ in a dedicated ontology. In our representation, each node of a graph is an item and has a score, which is computed through the consolidating function. Since we want to design an architecture that supports evaluating different implementations of this function, we introduce in each ontology of the architecture an upper-representation parametrising some shared semantic.

We design this representation since we do not want to force the perceiving and acting components of the architecture to synchronise with other components that would generate forgetting and consolidating stimuli. Therefore, we based the consolidating and forgetting functions based only on the data in the memory and internal stimuli that are automatically generated from components processing external stimulus, *i.e.* the storing and retrieving reasoners, which are synchronised with perceiving and acting modules.

Therefore, when the internal stimuli are triggered, the only source of knowledge for the internal reasoner is the memory M. Namely, the external reasoners are in charge to assure in the memory all the data required from the internal reasoners, before that the former triggers the later via an internal stimulus. Since we consider recurrent or demanded situations to affect the scores in memory, we suppose that the storing function $\mathcal{T}$ generates information to know when a situation as been re-experienced. While the retrieving function $\mathcal{R}$ should produce data in the ontology to know when a situation is recalled frequently. Remarkably, the consolidating function $\mathcal{N}$ computes the score also based on this data, so the upper-ontology presented below represents part of the semantic interface between the component computing $\mathcal{N}$ when it is triggered from $\mathcal{T}$ or $\mathcal{R}$.

In particular, the upper-ontology to represent frequently experienced or demanded situations include in the ontology two properties for each j-th items in memory. Those properties are meant for counting

(*i*) the number of times that an item has been experienced, *i.e.* the storing counter s_j, and (*ii*) the number of time that an item has been recalled, *i.e.* the retrieving counter r_j. The consolidating function is supposed to update the score of each item given those value and the structure itself of the memory, *i.e.* the graph of ontological classes Φ. After the computation performed by the consolidating functions, we assume the counters to be reset.

For using this representation in $\mathbb{X}$, we extended the semantic interface of SIT by adding in the ontology an individual ξ_j described with the data properties $\texttt{isRetrieved}(\xi_j, r_j)$ and $\texttt{isStored}(\xi_j, s_j)$. For each item in memory Φ_j, we consider the availability of an individual ξ, *i.e.* $\Phi_j \sqsubset \texttt{COUNTER}\colon \xi_j$, which is shared between the retrieving, storing, and consolidating functions.

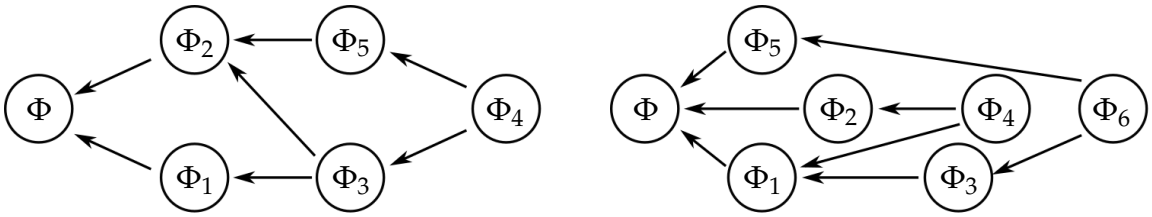
10.6.2 Upper Ontology for Item's Score

For each score u_j in a memory type, we introduced the forgetting function as a function that classifies items to have either a high or low or weak score, introduced in Section 10.4. The upper-ontology presented in this section is related to such a representation that we design for not limiting the other modules of the architecture similar to the previous section.

In particular, in this case, the retrieving function should assess the score level of each classified item and not return the ones with a low score. Indeed, the architecture is supposed to be able to retrieve them only when the consolidating function is such to increase their score to the high level. In contrary, scores that are considered to be weak are directly removed from the forgetting function, *i.e.* they cannot be retrieved anymore.

Therefore we extend the semantic interface of SIT for making the forgetting function able to add knowledge to the ontology that will be used by the retrieving function. With this purpose, we define an upper-ontology defining the concepts of items with a `HIGH`, `LOW`, or `WEAK` score. We define those symbols as three disjointed classes implying memory items, *i.e.* we identify an item with a low score when $\Phi_j \sqsubset \texttt{LOW}$, and analogously for high and weak levels. Remarkably, we do not limit the architecture to use logical reasoning for classifying the three score levels. Therefore, in the ontology, we consider axioms expressing the level of an item (*i.e.* $\sqsubset$) could also be asserted based on computations that are not performed through logic formalism.

To represent the numerical value of the score of each *j*-th item, we exploited the representation introduced in the previous section. In particular, we add a property `hasScore` to each individual ξ_j in order to relate a real number u_j with the item Φ_j, *i.e.* $\texttt{hasScore}(\xi_j, u_j)$ and $\Phi_j\colon \xi_j$.



(a) Items structure learned in the semantic memory.

(b) Items structure in the episodic memory.

Figure 10.5: The memory structure obtained from two instances of the SIT algorithm applied to the scenarios in Figure 10.4, and coupled with semantic interfaces specialised for the semantic and episodic memory. SIT computes the graphs based on the knowledge in the third column of tables 10.1 and 10.2 respectively.

10.7 Memory Funtionalities

In this section, we specify the implementation of all the functions introduced in Section 10.4 for a generic p-th memory type. While in sections 10.8 and 10.9 we will specify their behaviour for the semantic and episodic memory. In this section, we assume to have an ontology containing prior knowledge of some characteristics $\mathbf{R}$ and types Γ, which are used for specifying facts $f_i \in F$ in accordance with the semantic interface in Section 10.6.

10.7.1 Encoding

As mentioned we identify the encoding function $\mathcal{E}$ to be equivalent to the perceiving function defined in SIT. Namely,

$$\begin{aligned} \mathcal{E}(f_i) &= \mathscr{R}(\mathbf{R}_z, \gamma_x)\left(\epsilon, \gamma_y\right) && z, x, y : f_i \\ &= \mathbf{R}_{zx}\left(\epsilon, \gamma_y\right) \doteq b_r, && (10.5) \end{aligned}$$

which computes the beliefs of a situation based on facts. The beliefs are based on the reification $\mathscr{R}$ of the type of the instance involved in the domain of a characteristic Γ_x and the characteristic $\mathbf{r}_z$.

10.7.2 Storing

The storing function is implemented through a combination of the classifying $\mathcal{C}$, learning $\mathcal{L}$ and structuring $\mathcal{S}$ functions provided by SIT. Also, in accordance with Section 10.6.1, the storing function is in charge to update the state of each item in memory by increasing each related counter ξ_j.

Given the beliefs about an encoded situation $\epsilon = \mathcal{E}(F)$, we perform the classification function to identify if its beliefs can be represented

as items modelling a past situations. Practically, we perform a query in the ontology to find the classifications of ϵ in the memory Φ

$$\mathcal{C}(\Phi, \epsilon, \ell): \ \Phi_? \sqsubseteq \bar{\Phi} \iff \left(\Phi_? \sqsubseteq \Phi\right) \wedge \left(\Phi_? : \epsilon\right) \wedge \left({}^?d_\epsilon \geqslant \ell\right), \quad (10.6)$$

The query, solved with respect to $\Phi_?$ returns a graph of items $\bar{X}$ represented in the ontology as a hierarchy of classes $\bar{\Phi}$, whose features match the beliefs of ϵ with a similarity value bigger than a threshold ℓ, as presented in Chapter 7.

If the queried structure of items $\bar{\Phi}$ contains at least a node apart from the root class Φ, we consider the situation classified. In this case, for each j-th item that model the situation e (*i.e.* nodes of $\bar{\Phi}$), the storing function $\mathcal{T}$ increases the property `isStored` of the relative counter ξ_j in the ontology.

On the other hand, if no past items classify the situation (*i.e.* $\bar{\Phi}$ contains only the root Φ), we perform the learning function $\mathcal{L}$ to generate a new item Φ_ϵ form a situation ϵ

$$\mathcal{L}(\epsilon) = \Phi \prod_{zxy:x_i}^{i\in[1\ldots m]} \geqslant k_{zxy} \ \mathbf{R}_{zx}.\Gamma_y \doteq \Phi_\epsilon. \quad (10.7)$$

Since we adopted the open-word assumption, Φ_ϵ is a class with at most $m = v \cdot w \cdot w$ features, which are represented through a cardinality restrictions.

Φ_ϵ is added to the TBox of the ontology as a new class and, in accordance with the SIT algorithm, we require to update the OWL reasoner. The latter performs the structuring function to arrange the new item Φ_ϵ in the memory graph Φ. In this case, the score of the item is initialised to a neutral value, *i.e.* 0.5.

10.7.3 Retrieving

The retrieving function $\mathcal{R}$ is used to recall an item given the beliefs that – eventually partially – define a situation to be remembered. In this paper, we retrieve items through a *simulated* situation, which is not perceived directly from the environment but generated with the explicit purpose of remembering. In particular, we used dialogues aimed at specifying to the robot the knowledge to be encoded through $\mathcal{E}$ for retrieving an item. Therefore, through dialogue, the robot is supposed to understand a set of objects, with some characteristics, represented in accordance with the semantic interface used for the SIT instances. In other words, through dialogues, the user should provide the same type of information that the perception modules could generate. In this way, the facts perceived from the RGB-D camera or through dialogues would be consistent.

Given a situation ϵ encoded from a dialogue between the robot and the user. The classification function $\mathcal{C}$ can be used to retrieve a hierarchy of classes containing the items with features that matches the beliefs about the situation. Therefore if the user provides the robot with the beliefs that were stored in an item, the algorithm would return the items that match such a situation and all its parent and ancestors in the memory Φ. However, this is not the case if the user tells the robot only a subset of beliefs of the item to be retrieved.

For instance, if the user requests to retrieve a situation where "two spheres has two spheres on the right-hand side" from the memory structured in Figure 10.5a. Then, SIT will return a graph containing classified items structured as $\Phi_5 \sqsubset \Phi_2 \sqsubset \Phi$, where the similarity value with Φ_5 is equal to 1, *i.e.* it models perfectly the beliefs generated from the user sentence. While Φ_2 is a model describing only a part of the user's requests, *i.e.* it refers to a simpler item.

Nevertheless, if the user gives only a part of the facts that characterise the situation, *i.e.* "a sphere has a cone on the right-hand side". The classification function of SIT would not be able to classify any items. This occurs since the robot experiences depend on the situations in Figure 10.4, where each situation having a cone on the right-hand side of a sphere also involves other objects, *i.e.* a situation defined only with the beliefs $\texttt{rightSPHERE}(\epsilon, \texttt{CONE} : \gamma_y)$ do not respect any set of restrictions in the fourth column of Table 10.1.

In order to retrieve items less explicitly, we used the learning and structuring function of SIT instead of $\mathcal{C}$. In particular, given some beliefs encoded from a dialogue, it is possible to perform the learning function $\mathcal{L}$ to generate a dummy item, which is structured in the ontology through $\mathcal{S}$. Then, the memory Φ would contain a dummy node $\Phi_{\mathcal{R}}$, which represent a model of the situation that the robot should retrieve. With this process, we want to reason on the implications among situation, expressed as edges in Φ, in order to retrieve items Φ_j related to $\Phi_{\mathcal{R}}$. Then, we will remove from the ontology the dummy item $\Phi_{\mathcal{R}}$ for not affecting the robot's memory.

The SIT algorithm maintains a structure having a root Φ, representing an empty situation, and some leaves, which are the items modelling the most complex situations that the robot experienced, *i.e.* the item with the highest number of features. In the hierarchy, the children and successors of the dummy item, *i.e.* nodes $\Phi_j \sqsubset \Phi_{\mathcal{R}}$, identify models of situations where at least the features requested through dialogue are satisfied. Instead, the parents and ancestors of $\Phi_{\mathcal{R}}$, *i.e.* nodes $\Phi_j \sqsupset \Phi_{\mathcal{R}}$, are items modelling a part of the situation request by the user.

For instance, if the user request to remember a situation where "a SPHERE has a CONE and a SPHERE on the right-hand side", and "a CONE has a SPHERE on the right-hand side", *i.e.* a situation where a cone has a sphere on the right-hand side and another sphere on the

left-hand side; which have never been explicitly experienced by the robot from the situations in Figure 10.4. Then, the learning function would generate a dummy item $\Phi_{\mathcal{R}} \doteq \Phi \sqcap \geqslant 1$ `rightSPHERE.CONE` $\sqcap \geqslant 1$ `rightSPHERE.CONE` $\sqcap \geqslant 1$ `rightCONE.SPHERE`.

The structuring function would arrange the new item in such a way that the situations Φ_4 is a child of $\Phi_{\mathcal{R}}$, *i.e.* a more complex situation than the one requested from the user. While the parents of the dummy item would be Φ_3 and Φ_5 since both of their models could be consistently extended from $\Phi_{\mathcal{R}}$. Remarkably there could be the case in which the only parent of the dummy item is the root. This implies that the robot did not experience any simpler situation that can be described with the dummy item. Similarly, there could be the case where the dummy item does not have any children, which identifies that the robot never experienced such a complex situation.

Therefore it is possible to reduce the retrieving function to the storing function when the stored item is removed after having queried all the paths that connects the root and the leaves, passing through the dummy stored item $\Phi_{\mathcal{R}}$. The union of all those paths is a sub-graph Φ^*, which is the result of the retrieving function. In accordance with Section 10.6.1, for all the items in Φ^*, and the retrieving function is in charge to increase the `isRetrieved` value assigned to the relative counter ξ_j. Moreover, we introduce in Section 10.6.2 that items are classified in three levels of scores. The retrieving function should exclude from Φ^* the classes that do not have a high value, but it should increase the ξ_j counters also for the removed items. This is required to make the consolidating function increasing the score of a low item because it represent the model of a requested situation.

Remarkably, the storing function is suppose to make a preliminary classification attempt, this step could be used to assess if the user provides the robot with an exhaustive set of beliefs. Indeed, since the structuring function is the most complex computation of SIT, the classification might reduce the computation time if the user gives a precise set of facts to the robot.

10.7.4 Consolidating

As mentioned, we want to give an expressive interface for implementing the consolidating function, and with this purpose, we provide it with full access to the memory. In particular, this function is supposed to be performed for each j-th item in memory and should be such to update their scores ξ_j. Therefore, it needs to replace the value of the score in the ontology, and it can base this operation on some parameters.

Those parameters can be based on the counters s_j and r_j, which represent the number of times that each item has been stored or retrieved. Also, it can rely on the items cardinally ${}^j d$, which measures the differ-

Situation	Facts	Beliefs	Item
1	CONE: γ_1, SPHERE: γ_2, right(γ_1, γ_2).	rightCONE(ϵ, γ_2).	$\Phi_1 \doteq \Phi \sqcap \geqslant 1$ rightCONE.SPHERE.
2	SPHERE: γ_3, γ_4, right(γ_3, γ_4).	rightSPHERE(ϵ, γ_4).	$\Phi_2 \doteq \Phi \sqcap \geqslant 1$ rightSPHERE.SPHERE.
3	CONE: γ_5, SPHERE: γ_6, γ_7, right(γ_5, γ_6), right(γ_6, γ_7), right(γ_5, γ_7).	rightCONE(ϵ, γ_6), rightCONE(ϵ, γ_7), rightSPHERE(ϵ, γ_7).	$\Phi_3 \doteq \Phi \sqcap \geqslant 1$ rightSPHERE.CONE $\sqcap \geqslant 2$ rightCONE.SPHERE.
4	CONE: γ_9, SPHERE: γ_8, γ_{10}, γ_{11} right(γ_8, γ_9), right(γ_8, γ_{10}), right(γ_8, γ_{11}), right(γ_9, γ_{10}), right(γ_9, γ_{11}), right$(\gamma_{10}, \gamma_{11})$.	rightCONE(ϵ, γ_{10}), rightCONE(ϵ, γ_{11}), rightSPHERE(ϵ, γ_9), rightSPHERE(ϵ, γ_{10}), rightSPHERE(ϵ, γ_{11}).	$\Phi_4 \doteq \Phi \sqcap \geqslant 1$ rightSPHERE.CONE $\sqcap \geqslant 2$ rightCONE.SPHERE $\sqcap \geqslant 2$ rightSPHERE.SPHERE.
5	SPHERE: γ_{12}, γ_{13}, γ_{14}, right$(\gamma_{12}, \gamma_{13})$, right$(\gamma_{12}, \gamma_{14})$, right$(\gamma_{13}, \gamma_{14})$.	rightSPHERE(ϵ, γ_{13}), rightSPHERE(ϵ, γ_{14}).	$\Phi_5 \doteq \Phi \sqcap \geqslant 2$ rightSPHERE.SPHERE.
6	CONE: γ_{16}, SPHERE: γ_{15}, γ_{17}, γ_{18} right$(\gamma_{15}, \gamma_{16})$, right$(\gamma_{15}, \gamma_{17})$, right$(\gamma_{15}, \gamma_{18})$, right$(\gamma_{16}, \gamma_{17})$, right$(\gamma_{16}, \gamma_{18})$, right$(\gamma_{17}, \gamma_{18})$.	rightCONE(ϵ, γ_{17}), rightCONE(ϵ, γ_{18}), rightSPHERE(ϵ, γ_{16}), rightSPHERE(ϵ, γ_{17}), rightSPHERE(ϵ, γ_{18}).	$\Phi_4 : \epsilon$

Table 10.1: The knowledge in the semantic memory for each situation in Figure 10.4, which are experienced by the robot with the presented order.

ence between items complexity, as detailed in Section 7.5.5. Moreover, it could use the scores of the items that are parents or children of the considered item Φ_j to consolidated, *i.e.* it can explore the memory and access the representation it contains through logic queries. Indeed there are many heuristics to be validated and developed, and our architecture is mainly design for investigating them. Therefore, we do not provide in this paper a general definition of the consolidating function, and for implementing the architecture we assume only that the this function changes the hasScore value assigned to the ξ_j individual of each Φ_j item. However, for our preliminary scenario, we present a possible approach for the semantic and for the episodic memory in sections 10.8 and 10.9 respectively.

10.7.5 Forgetting

We define in Section 10.4 the forgetting function as the operator that classifies items based on their score u_j. We suppose the score to be classified in three levels, *i.e.* high, low and weak. In accordance with the representation presented in Section 10.6.2, the forgetting function is in charge to access the score of each item ξ_j and, based on it, it classifies

each item to be in a score level. If there are any, the forgetting function removes the weak items from memory, while the differences between high and low levels would be used during the retrieving process as discussed in Section 10.7.3. Remarkably, after having removed an item from the graph, its edges should be recomputed through the $\mathcal{S}$ function, which is based on OWL reasoning.

In our simplified scenario, we implemented this classification based on two thresholds in [0,1], which identify three disjoint ranges. Nevertheless, we design the architecture to support also different implementations of this function, and we aim to exploit it in order to evaluate more sophisticated solutions. However, we argue that a well posed consolidating function would generate more effective forgetting policy than a complex method to classify items based on their score, if the latter are not represented appropriately.

10.8 Semantic Memory

In the referencing scenario, we set the SIT instance delegated to the maintain the semantic memory with an interface similar to the one used in Chapter 7. In particular, we consider a SIT instance based on prior knowledge in the ontology specialised for the purposes of (*i*) classifying instance as object with a geometrical shape, *i.e.* $\Gamma = \{\texttt{SPHERE}, \texttt{CONE}\}$, and (*ii*) representing characteristic of facts as qualitative spatial relations, *i.e.* $\mathbf{R} = \{\texttt{right}\}$. Remarkably, in this paper, we reduce the set of possible shapes and spatial relation only for demonstration purposes, but our method does not limit the number of characteristics v and types w. Nevertheless, those values affect the complexity of the algorithm as detailed in Section 7.6.6.

Given this semantic, interface SIT structures items regarding objects with an abstract type and spatial relations that do not depend on the context, but only relates pairs of objects independently. We adopted this formalism with the purpose to highlight the features and limitations of the architecture, but a more sophisticated representation of semantic items are undoubtedly desirable. As discussed in Chapter 7, since the algorithm is based on a purely symbolical representation it is possible to change the semantic of beliefs and features through different interfaces without affecting the architecture (we provide an example of this behaviour of SIT in Section 10.9).

As an example, consider the situations shown in Figure 10.4, the Table 10.1, and Figure 10.5a. The former figure shows situations that are demonstrated to the robot in sequence over time. For each of those, it is assumed that architecture generates input facts as shown in the second column of the table. The third column of the table shows beliefs of each situation that we encode based on Equation 10.5, while the fourth column shows the relative learned item that we stored based on Equation 10.7. In this example, we assume that each situation is

Situation	Facts	Beliefs	Item
1	TABLE1: γ_l, TIME: γ_t, EPISODIC: γ_E, CONE RED BIG: γ_1, SPHERE GREEN BIG: γ_2, hasLOCATION(γ_E, γ_l), hasTIME(γ_E, γ_t) has(γ_E, γ_1), has(γ_E, γ_2).	hasEPISODIC(ϵ, γ_l), hasEPISODIC(ϵ, γ_t), hasEPISODIC(ϵ, γ_1), hasEPISODIC(ϵ, γ_2).	$\Phi_1 \doteq \Phi_E \sqcap$ $\geqslant 1$ hasEPISODIC.LOCATION $\sqcap$ $\geqslant 1$ hasEPISODIC.TIME $\sqcap$ $\geqslant 1$ hasEPISODIC.CONE RED BIG $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE GREEN BIG.
2	TABLE1: γ_l, TIME: γ_t, EPISODIC: γ_E, SPHERE RED SMALL: γ_3, SPHERE BLUE BIG: γ_4, hasLOCATION(γ_E, γ_l), hasTIME(γ_E, γ_t) has(γ_E, γ_3), has(γ_E, γ_4).	hasEPISODIC(ϵ, γ_l), hasEPISODIC(ϵ, γ_t), hasEPISODIC(ϵ, γ_3), hasEPISODIC(ϵ, γ_4).	$\Phi_2 \doteq \Phi_E \sqcap$ $\geqslant 1$ hasEPISODIC.LOCATION $\sqcap$ $\geqslant 1$ hasEPISODIC.TIME $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE RED SMALL $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE BLUE BIG.
3	TABLE1: γ_l, TIME: γ_t, EPISODIC: γ_E, CONE RED BIG: γ_5, SPHERE BLUE SMALL: γ_6, SPHERE GREEN BIG: γ_7, hasLOCATION(γ_E, γ_l), hasTIME(γ_E, γ_t) has(γ_E, γ_5), has(γ_E, γ_6), has(γ_E, γ_7).	hasEPISODIC(ϵ, γ_l), hasEPISODIC(ϵ, γ_t), hasEPISODIC(ϵ, γ_5), hasEPISODIC(ϵ, γ_6), hasEPISODIC(ϵ, γ_7).	$\Phi_3 \doteq \Phi_E \sqcap$ $\geqslant 1$ hasEPISODIC.LOCATION $\sqcap$ $\geqslant 1$ hasEPISODIC.TIME $\sqcap$ $\geqslant 1$ hasEPISODIC.CONE RED BIG $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE BLUE SMALL $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE GREEN BIG.
4	TABLE1: γ_l, TIME: γ_t, EPISODIC: γ_E, SPHERE GREEN SMALL: γ_8, CONE RED BIG: γ_9, SPHERE BLUE BIG: γ_{10}, SPHERE GREEN BIG: γ_{11}, hasLOCATION(γ_E, γ_l), hasTIME(γ_E, γ_t) has(γ_E, γ_8), has(γ_E, γ_9), has(γ_E, γ_{10}), has(γ_E, γ_{11}).	hasEPISODIC(ϵ, γ_l), hasEPISODIC(ϵ, γ_t), hasEPISODIC(ϵ, γ_8), hasEPISODIC(ϵ, γ_9), hasEPISODIC(ϵ, γ_{10}), hasEPISODIC(ϵ, γ_{11}).	$\Phi_4 \doteq \Phi_E \sqcap$ $\geqslant 1$ hasEPISODIC.LOCATION $\sqcap$ $\geqslant 1$ hasEPISODIC.TIME $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE GREEN SMALL $\sqcap$ $\geqslant 1$ hasEPISODIC.CONE RED BIG $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE BLUE BIG $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE GREEN BIG.
5	TABLE1: γ_l, TIME: γ_t, EPISODIC: γ_E, SPHERE BLUE SMALL: γ_{12}, SPHERE GREEN BIG: γ_{13}, SPHERE BLUE SMALL: γ_{14} hasLOCATION(γ_E, γ_l), hasTIME(γ_E, γ_t) has(γ_E, γ_{12}), has(γ_E, γ_{13}), has(γ_E, γ_{14}).	hasEPISODIC(ϵ, γ_l), hasEPISODIC(ϵ, γ_t), hasEPISODIC(ϵ, γ_{12}), hasEPISODIC(ϵ, γ_{13}), hasEPISODIC(ϵ, γ_{14}).	$\Phi_5 \doteq \Phi_E \sqcap$ $\geqslant 1$ hasEPISODIC.LOCATION $\sqcap$ $\geqslant 1$ hasEPISODIC.TIME $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE GREEN BIG $\sqcap$ $\geqslant 2$ hasEPISODIC. SPHERE BLUE SMALL.
6	TABLE1: γ_l, TIME: γ_t, EPISODIC: γ_E, SPHERE BLUE SMALL: γ_{15}, CONE RED BIG: γ_{16}, SPHERE BLUE SMALL: γ_{17}, SPHERE GREEN BIG: γ_{18}, hasLOCATION(γ_E, γ_l), hasTIME(γ_E, γ_t) has(γ_E, γ_{15}), has(γ_E, γ_{16}), has(γ_E, γ_{17}).	hasEPISODIC(ϵ, γ_l), hasEPISODIC(ϵ, γ_t), hasEPISODIC(ϵ, γ_{15}), hasEPISODIC(ϵ, γ_{16}), hasEPISODIC(ϵ, γ_{17}), hasEPISODIC(ϵ, γ_{18}).	$\Phi_6 \doteq \Phi_E \sqcap$ $\geqslant 1$ hasEPISODIC.LOCATION $\sqcap$ $\geqslant 1$ hasEPISODIC.TIME $\sqcap$ $\geqslant 2$ hasEPISODIC.SPHERE BLUE SMALL $\sqcap$ $\geqslant 1$ hasEPISODIC.CONE RED BIG $\sqcap$ $\geqslant 1$ hasEPISODIC.SPHERE GREEN BIG.

Table 10.2: The knowledge in the episodic memory for each situation in Figure 10.4, which are experienced by the robot with the presented order.

never classified in another during the storing operation. Therefore all the items would have an initialised score value and, at the end of all the demonstrations, they are structured in an graph as shown in the figure. Remarkably, in the following sections we discuss how a different semantic interface can be used to process the same situations to obtain knowledge shown in Table 10.2, which is structured in a different graph that is shown in Figure 10.5b.

10.8.1 Semantic Score

In our scenario, we implement the consolidating function such to compute the score of each j-th items in the semantic memory as a weighted sum among (i) its complexity ${}^{j}d$, defined as the sum of all the features of an item (Equation 10.4), (ii) the number of children in the implication graph (n_c) and, if it $n_c > 0$, their score (ξ_i, $i \in [1, n_c]$), as well as (iv) the number of times that Φ_j has been stored s_j and (v) retrieved r_j, which are related in the individual ξ_j in accordance with Section 10.6.1.

Formally, we define the reinforcing function as

$$\bar{\xi}_j = a_d {}^{j}d + a_c \frac{\sum_i \xi_i}{n_c} + a_s s_j + a_r r_j, \tag{10.8}$$

where all the weights $a_d, a_c, a_s, a_r \in \mathbb{R}^+$. From the equation we observe that $\bar{\xi}_i$ depends from ${}^{j}d$ which does not change over time. Its contributions penalises complex items which might be related to specific experiences, against simpler items. On the other hand, the other contributions of the weighted sum are dynamic, and we observe that the score of the j-th item increases *iff.* (i) the average score of more complex items (the i-th children) increases, as well as if (ii) the j-th item has been stored or (iii) retrieved again. In particular, the i condition implies that, given $\Phi_j \sqsubset \Phi_i$, the score of Φ_j increases if the score of Φ_i increases; thus Equation 10.8 tends to give importance to the simpler items related to important items.

Generally, the computation of the score above might generate any positive numbers. For this reason, it is not possible to compare $\bar{\xi}_i$ among the items arranged in a general implication graph, or among graphs representing different memory types. Therefore, after the computation of Equation 10.8 for all items in the ontology, we normalise the score in [0,1]. Namely, we formalise the reinforcing function for the instance of SIT delegated to represent the semantic memory in our scenario as

$$\mathcal{N} : \xi_j = \frac{\bar{\xi}_j}{\sum_{\Phi_j \sqsupseteq \Phi} \bar{\xi}_j}, \tag{10.9}$$

where such a new score ξ_j overwrites the old score of the j-th item only in the ontology specialised for the semantic memory.

10.9 Episodic Memory

Smartly to the semantic memory, for the episodic structure, we use a specific semantic interface for configuring SIT. In this case, we consider more complex prior knowledge for spanning $\mathbf{R}$ and Γ, which requires some preliminary computation for generating processable facts. The latter, for episodic items, are supposed to be represented through a temporal reference, a location, and a encoding of the situation itself. For reasoning on this knowledge we need to adapt the semantic interface of SIT.

For doing so, we assume that any situation ϵ in the episodic memory contains a dummy instance, *i.e.* $\texttt{EPISODIC}:\gamma_E$. The later is supposed to be involved always in only two facts through the characteristics `hasLocation` and `hasTime`, *e.g.* $\texttt{hasTime}(\gamma_E, 12{:}45)$. Specifically for the locations, we use symbols to discriminate tables, *e.g.* $\texttt{hasLocation}(\gamma_E, \gamma_y)$, where $\texttt{TABLE1}:\ \gamma_y$. In this way we encode all the episodic situation through the belief $\texttt{hasLocationEPISODIC}(\epsilon, \gamma_y)$. Therefore, each episodic item in the memory graph will be a child of a class defining time and location, *e.g.* $\Phi_E = \Phi \sqcap \geqslant 1\ \texttt{hasLocationEPISODIC.TABLE1} \sqcap \geqslant 1\ \texttt{hasTimeEPISODIC.Time}$. Remarkably, with this representation, if the same situation is shown on two different tables, they would be modelled as two independent items, *i.e.* nodes that are not connected between each other in the memory Φ. This does not occur for the representation of time since we consider a unique type `TIME` for all the possible instances γ_y.

We represent episodic types Γ not only with their abstract shape but also with characteristic proper of each object, such as the colour and dimension. In particular, the colours are represent in classes, *i.e.* `RED`, `GREEN`, and `BLUE`, as well as the dimensions, *i.e.* `BIG` and `SMALL`. To keep into account three different semantic as shape, colour and dimension and represent the type of an instance unambiguously, we reify them for all their combinations in order to obtain the set of types to be given as prior knowledge, *i.e.* $\Gamma = \{\texttt{SPHERE RED BIG}, \texttt{SPHERE RED SMALL}, \texttt{SPHERE GREEN BIG}, \ldots, \texttt{CONE BLUE SMALL}\}$.

For applying this semantic interface consistently, we require again the dummy instance γ_E, which is considered to be related to the objects on the table through the characteristic `has`. Thus, we generate facts of the type $\texttt{has}(\gamma_E, \gamma_y)$, where $\texttt{CONE RED BIG}:\ \gamma_y$ for instance. Given this fact f_i, Equation 10.5 would compute a belief $\texttt{hasEPISODIC}(\gamma_E, \gamma_y)$, which might generate a restriction like $\geqslant 1\ \texttt{hasEPISODIC.CONE RED BIG}$. Therefore the set of characteristics we used for the semantic interface is $\mathbf{R} = \{\texttt{has}, \texttt{hasLocation}, \texttt{hasTime}\}$.

Table 10.2 shows the computation performed by SIT with this semantic interface for the same situation considered in the previous section, *i.e.* Figure 10.4. In this case, after the architecture stores all the situations, the structure shown in Figure 10.5b is available in the

episodic memory of the robot. From the comparison with the semantic memory, we can observe that the two structures are different even if generated from the same situation. Indeed, differently from the previous instance of SIT, in this case, spatial relation does not affect the representation of items since the latter are represented only through features based on the characteristics of each object. Although our representation might not be accurate for representing semantic items in a real scenario, with this example, we have shown that SIT can be used to evaluate complex semantic without affecting the other parts of the architecture.

10.9.1 Episodic Score

Since we design the episodic and semantic ontologies based on a common structure, we can define the reinforcing function for computing the episodic score of an item similar to Section 10.8.1 but, since the two ontologies have represented memory graphs differently, the ranking behaviour would be different. In this section we extend the weighted sum in Equation 10.8 for the episodic memory, which would be used for the relative consolidating function to compute the score of each j-th items.

In particular, for computing the episodic scores we introduce the instant of time t_j in which the j-th item has been stored the last time. This implies that the storing function of the SIT instance dedicated to the episodic memory should update the time instance of an item when it is classified, and not only update its storing counter as the semantic storing function does. Noteworthy, similar behaviour could also be considered for the retrieving time.

In the weighted sum introduced for computing semantic scores, we also considered the item's complexity ${}^{j}d$. For the episodic score, the latter will not count the spatial relations in a situation as in the semantic memory anymore. Instead, it will count the total numbers of objects having the same shape, colour and dimension. However, the complexity measure of episodic items can still be computed through Equation 10.4 applied to the ontology managed from the SIT instance devoted to maintaining the episodic memory.

Moreover, also the component of the score depending on the related items changes since we used a different semantic interface. In particular, in this case, each i-th children of an item Φ_j represents situations involving at least objects with the same characteristics, independently from their spatial arrangements. Therefore, by applying the same ranking policy of the semantic memory, *i.e.* Equation 10.8, we would weight the score of an item based on the importance that is given to other items involving the same objects.

Also, in the computation of the scores, we consider the storing and retrieving counters values as in the case of the semantic memory.

However, the episodic counters would have a different evolution with respect to the counters stored in the semantic memory, since the two graphs (figures 10.5a and 10.5b) are based on beliefs differently encoded from the same facts.

In summary, we define the computation of the episodic score as

$$\bar{\xi}_j = o_d{}^j d + o_c \frac{\sum_i \xi_i}{n_c} + o_s s_j + o_r r_j + o_t \log\left(t_0 - t_j\right), \tag{10.10}$$

where $o_d, o_c, o_s, o_r, o_t \in \mathbb{R}^+$, and the score ξ_j is computed through normalisation in [0,1] as in Equation 10.9. Noteworthy, in this implementation we consider score that temporally decay based on the last storing time t_j referred to an initial instant t_0 and, since the difference in timestamps would be much higher than all the other contributions of $\bar{\xi}_j$, we decided to use a logarithmic function.

Remarkably, from a long-term perspective, this definition of the scores rewards the differences among scores over time, because we normalise all the scores at each instant. Moreover, it is important to notice that since the semantic and episodic reinforcing functions (equations 10.8 and 10.10) do not generate a normalised value, a change of the score of a single item implies to update the scores of all the items in the graph. This might be an issue toward some real applications, but we argue for a large room of improvement in the definition of the function to compute the scores, and we believe our architecture gives suitable tools for systematically evaluate different methodologies.

10.10 Memory Managment

The previous section showed the structure of items that is possible to obtain for representing different memory types through the SIT algorithm. We observed that, based on the same experiences, SIT instances with different semantic interface create different memory graphs in the two ontologies, *i.e.* the items are represented with different perspectives. Nevertheless, those share also a common structure, since the computation involved in the same function of different memory types is analogous, *i.e.* they share most of the implementation. This section describes an architecture which uses the same design pattern for implementing the semantic and episodic memory based on the functions introduced in Section 10.7. In particular, we present an overview of all the proposed architecture, and we discuss the behaviour of the $\mathcal{E}$, $\mathcal{T}$, $\mathcal{R}$, $\mathcal{N}$, $\mathcal{F}$, for some situation processed in the semantic and episodic memory introduced in sections 10.8 and 10.9.

Figure 10.6 shows the architecture through interfaces between modules, in accordance with the component diagram style of the Unified Modeling Language (UML). We deploy our architecture in the Robot Operative System (ROS) which involves nodes and services

communicating messages through interfaces. A *node* is a component that communicates with the publish-subscribe messaging pattern (an asynchronous communication), and we identify its interfaces in the diagram with a single ball-socket symbol, which indicates the provided (the ball) or requested (the socket) message. On the other hand, a *services* is used from a client that establishes a request-response messaging pattern (synchronous communication), that we denote in the diagram with a double ball-socket symbol, indicating the type of the request message to provide to the server (light ball), and response it provides (dark ball).

The core of the architecture is the Knowledge Representation (KR), which contains the ontologies that contain most of the information processed from the architecture; therefore it is interfaced with most of the other components. To assure knowledge consistency in the architecture, we need to orchestrate the components that use the ontologies, while being responsive to external stimulus. We used the synchronous and asynchronous interfaces provided by ROS for managing the synchronisation among the components of the architecture, and Figure 10.7 shows the temporal behaviour of our design.

More in details, the figure shows the UML temporal diagram associated with the activation of the components and some highlights of the message exchanged through the interfaces of the architecture. In this diagram each column represents a component with respect to time, a bold vertical line represents the activation of each component, while a horizontal arrow identifies the sending of a message. As presented in Section 10.3, in our architecture messages are mainly triggering signals among components that consume and produce knowledge in the ontology with a specific format. Therefore, in the arrows of Figure 10.7 are mainly annotated not the data that is directly exchanged among components, but the knowledge that a component generates or queries from the ontology before to trigger other components, which are supposed to relay in the ontology for their inputs.

10.10.1 Architecture

We design the architecture through a pattern that allows for implementing the management of a generic memory type. Indeed, it is important to notice in Figure 10.6 that all the components that implement the functions introduced in Section 10.4 are duplicated for the semantic and episodic memory, but they mutually have the same interfaces and share most of the implementation, but with different semantics.

Moreover, it is possible to notice another design pattern we used, which assumes that the components requiring the knowledge representations (identified in red in the diagram) are reasoners (in green and yellow), which are interfaced among each other, as well as with

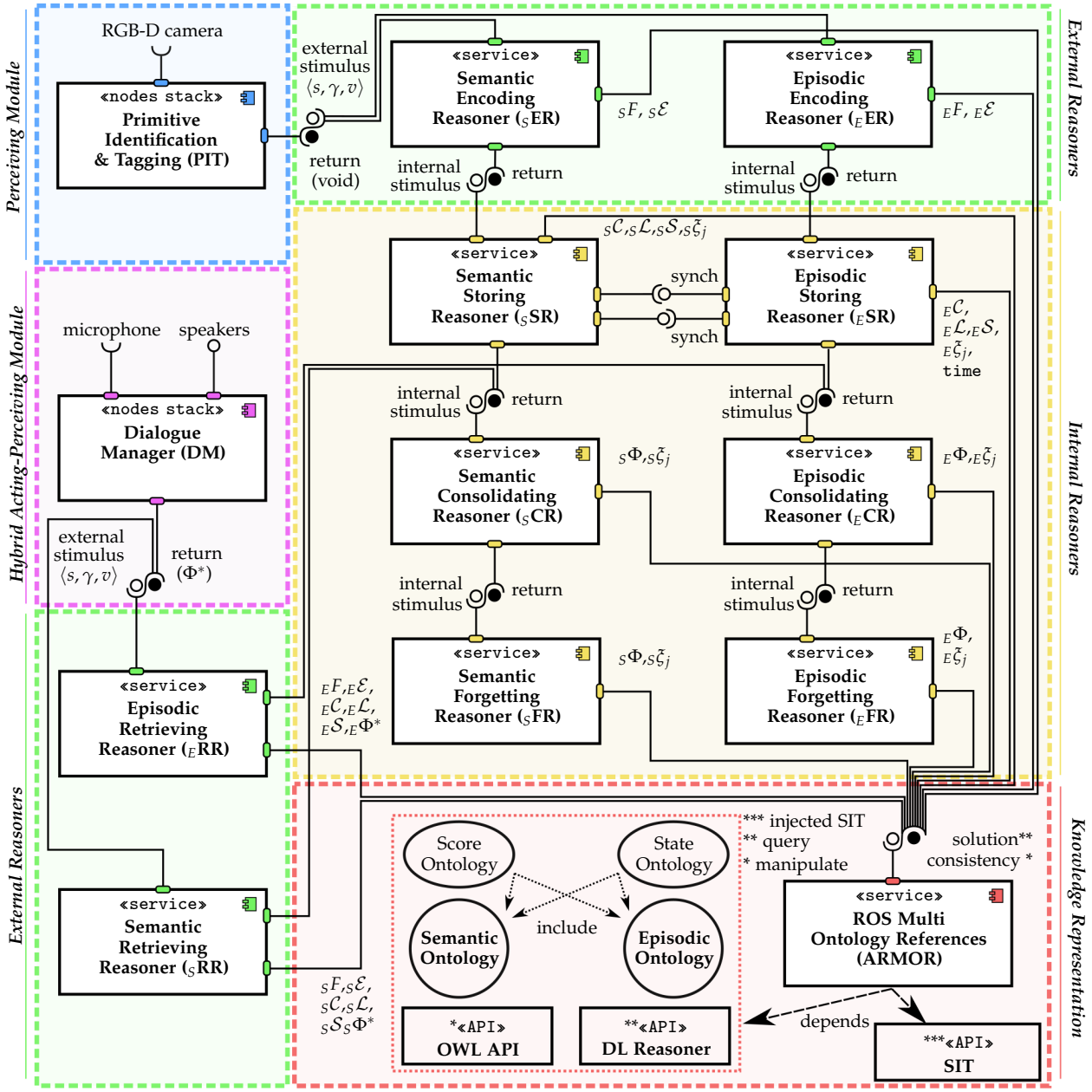


Figure 10.6: The software architecture where we deploy our design patter for managing the semantic and the episodic memory.

perceiving and acting modules (in blue). We design each reasoner to be a service that manipulates the knowledge representation, and we further identify them in two different types. An *external* reasoner (shown in green) is triggered from external stimuli (*i.e.* RGBD images or signals from the microphone). They are the only component of the architecture that receives data directly from another component, and have the purposes of introducing knowledge in the ontology as well as generate internal stimuli that trigger *internal* reasoners (shown in yellow). Each internal reasoner uses only the knowledge already

available in the ontologies and might generate further internal stimuli to other internal reasoners (*e.g.* the storing reasoner triggers the consolidating reasoner).

In our scenario, the architecture requires an RGB-D camera, whose data is processed by the Primitive Identification and Tagging (PIT) component Chapter 2, through which we detect objects on tables, and we extract some of their characteristics, *i.e.* position, shape, colour, and dimension. On the other hand, we interact with the user through the Dialogue Manager (DM) as presented in Chapter 9, where we use the same implementation of the semantic memory, presented in Section 10.8 to explain experiences to a supervisor, which could correct the beliefs of the robot.

Remarkably, in our scenario, the robot does not perform any physical actions, but our architecture can accommodate *acting* components as well. The limitation that the architecture might pose for planning and acting components is mainly due only to computational complexity, especially if raw measurements are used as feedback. As far as dialogues are concerned, we design the architecture with a *hybrid* perceiving-acting module (shown in purple in Figures 10.6). Thorough components, the robot understands user's sentences with retrieving purposes (*i.e.* perception), and it generates the explanation of retrieved memory items (*i.e.* action).

For making the ontologies available to distributed components in a ROS architecture we used a ROS Multi Ontology References (ARMOR) service Chapter 6. The latter uses OWL API [73] and a DL reasoner (we used Pellet [144]) to expose three types of services: (*i*) the manipulation of the knowledge that returns a consistency signal, (*ii*) the query to the ontology, which returns knowledge that solves it, and (*iii*) the functions provided by SIT, *i.e.* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{S}$, which are implemented in ARMOR as injected services.

Since ARMOR contains the memory and the functions that are semantically interfaced, the perceiving and acting modules should be designed in order to provide or obtain data with a well defined semantic, *e.g.* the semantics introduced in sections 10.8 and 10.9 for our scenario. In other words, each Encoding Reasoner (ER) requires data to generate feasible facts and beliefs about a situation in the relative ontology. Then, ER generates an internal stimulus that triggers the relative Storing Reasoner (SR). The latter might add a new item ontology or change the score of an item already in the memory. Since those changes affect the items scores, ER triggers the Consolidating Reasoner (CR), which uses this information to update the values of the scores. Consequently, CR triggers the Forgetting Reasoner (FR), which classifies the items based on the new score level, and deletes memories that have a weak score.

We assume that the above process driven from visual perception is triggered by the user, who might want the robot to observe a new

situation or to be involved in a retrieving task. The latter case implies a dialogue, and the understood information should be evaluated in the memory. The Retrieving Reasoner (RR) performs this operation through the services injected in ARMOR that implement SIT functionalities (Section 10.4.1). After, RR returns two graphs containing semantic and episodic items that have been retrieved and explained to the user.

10.10.2 Encoding Reasoners

The encoding function $\mathcal{E}$ (Equation 10.5) has the purpose of generating in the ontology facts and beliefs according the semantic interfaces, which are defined for the two memory types through sets of symbols $({}_S\mathbf{R},{}_S\Gamma)$ and $({}_E\mathbf{R},{}_E\Gamma)$ discussed in sections 10.8 and 10.9. For doing so, the perceiving modules, *i.e.* PIT, calls the episodic and semantic encoding reasoners (${}_E$ER and ${}_S$ER), with the purposes to compute the encoding function $\mathcal{E}$, which is implemented by a specific SIT instance injected in ARMOR. Since the two encoding reasoners exploit different semantic interface, the related ontologies would contain knowledge as shown in the third column of tables 10.1 and 10.2 respectively.

In accordance with our formalisation of SIT, the facts to encode can also be based on logic reasoning, *e.g.* to evaluate spatial relationships based on objects' position. For a general semantic interface, inputs that PIT provides to ER are a set of tuples of the type $\langle s, \gamma, v\rangle_i$, where s is a symbol semantically represented in the ontology, *e.g.* `hasColour`. While γ is an instance and v is a value, which can be a number or a symbol used for further reasoning, as well as another instance, *i.e.* PIT might already provide a fact f_i. Remarkably, in Chapter 4 we present a multimodal perceiving module that extends the perceiving components considered in this paper with more expressive and reliable information. Moreover, in the case in which the fuzzy implementation of SIT is considered then the facts and beliefs would also have another field, *i.e.* a fuzzy membership.

ER is an external reasoner, *i.e.* a ROS service, which receives input tuples and uses ARMOR for generating facts in an ontology. When the ontology contains all the knowledge for the current situation based on a semantic interface, ER uses SIT to compute the related beliefs, which are stored in the ontology. In Figure 10.7, those two operations are simplified as only one message sent at lines 2 and 3 (for ${}_SER$ and ${}_EER$ respectively). Those messages are sent as a reaction to knowledge provided by PIT aperiodically since we consider the user to arrange the situation and give an explicit command to the robot in order to observe it through the perceiving module. When the requests of ER to AMOR are performed (lines 4 and 5), ER continues its computations and triggers SR in order to evaluate the storing function for the new beliefs, *i.e.* it generates an internal stimulus shown at lines 6 and 7.

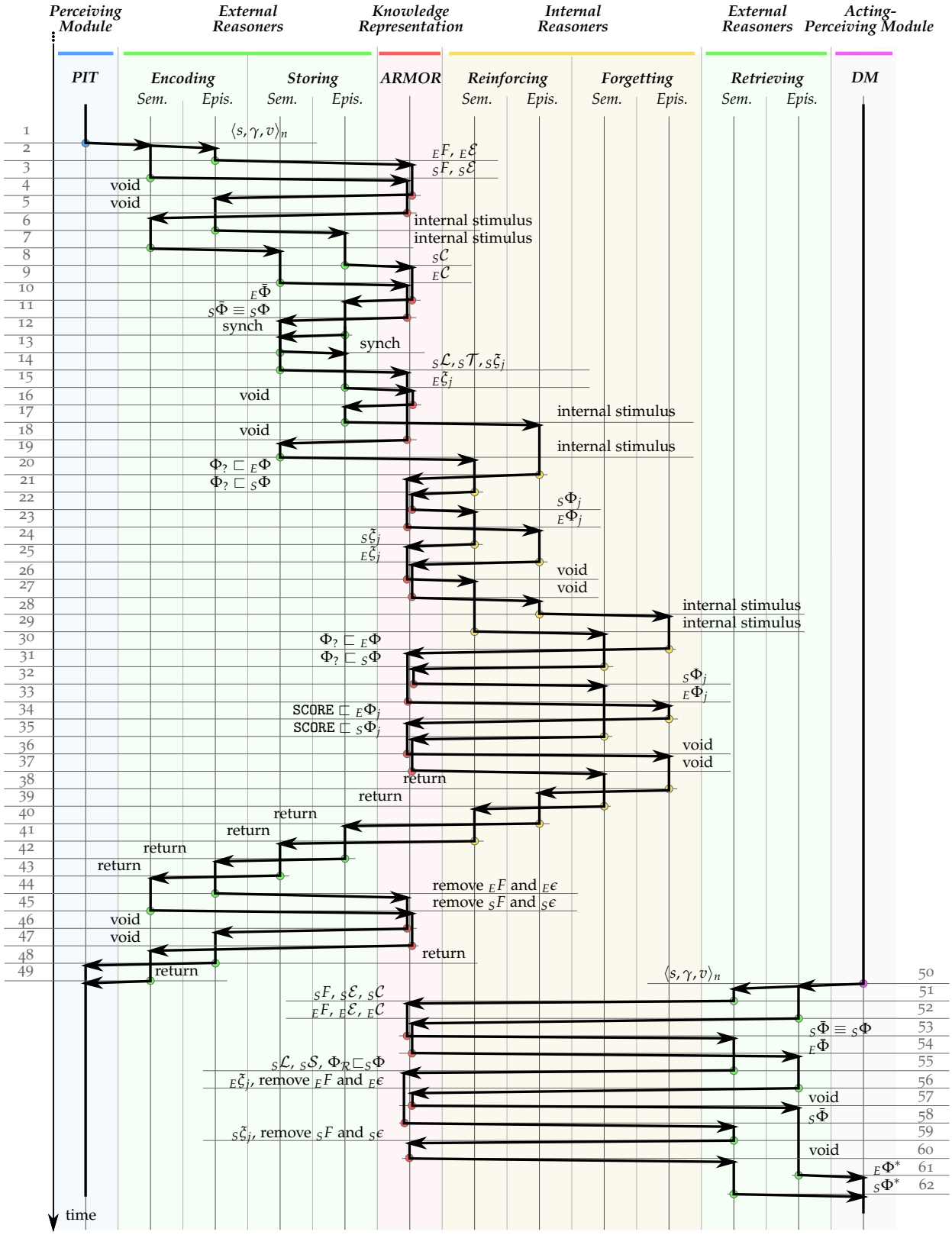


Figure 10.7: The activation of the components of the architecture in Figure 10.6, and the highlights of the exchanged messages as well as knowledge manipulations or queries.

Since also SR is ROS services, the first waits until the second terminates. After (at lines 44 and 45), ER continues its computations and removes all the facts and beliefs from the ontology (lines 46 and 47). Such a cleaning operation is required from SIT since it supposes the ontology to contain only facts about a single instant of time. This drawback also limits the computation of the retrieving function, which cannot be performed in parallel to ER and SR. Than, ER terminates (at lines 48 and 49) and the architecture might be used to process the point cloud of another situation.

10.10.3 Storing Reasoners

In Section 10.7.2, we discussed the implementation of the storing function, which is based on a combination of $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{S}$ functions provided by SIT. Between lines 8 and 19 of Figure 10.7, the semantic and episodic SR use ARMOR for performing those function assuming that the ER generated in the ontology beliefs about the situation to process. The storing function is such to increase the counters of an item through classification, or it would generate a new item in the memory, and in the figure, we show an example where ${}_S\mathcal{T}$ generates a new memory item, while ${}_E\mathcal{T}$ classifies a previous experience.

In the architecture, we store items for the semantic and episodic memory with independent instances of SIT. Nonetheless, it is possible to observe that the nodes in the figures 10.5a and 10.5b can have a name related to the situation, which can be used to associate items from independent graphs. In this way, it is possible to associate a semantic and an episodic item to the same situation, *i.e.* ${}_E\Phi_1$ and ${}_S\Phi_1$ describe the same situation that we named `Item1`. In our scenario, we generate sequential names, *e.g.* `Item1`, `Item2`, ..., but we discussed in Chapter 7 that meaningful names should be used for a real application, and in Chapter 9 we have shown a technique to involve a person for assigning names to items Φ_j.

However, due to the forgetting functions, which might remove an item only form some memory types, the associativity match could be not always possible. To overcome this problem, we introduce a synchronisation signal between ${}_S$SR and ${}_E$SR, as shown in Figure 10.6. In particular, the synchronisation signal is sent from a component to the others after the computation of $\mathcal{C}$. The synchronisation allows assuring that an eventually learning phase of SIT is performed by any SR after that all of them have terminated the first classification attempt. In this way, if one SR should learn a new item via $\mathcal{L}$ and $\mathcal{S}$, it could name the new item with the same name obtained in other memories through classification. The latter name can be accessed from the ontologies through ARMOR, but we accept it as a feasible name only if the classification has a similarity value ${}^jd_\epsilon = 1$, *i.e.* the features of an item perfectly model the belief of a situation. Only with this

condition, it is possible to maintain a consistent association between items, as well as their children and parents, among different graphs.

As an example, at lines 8 and 9 of Figure 10.7 the classification is performed through ARMOR, which returns a graph of items that classify the beliefs in the ontology. For the episodic memory, line 10, some items are given, while for the semantic memory, line 11, ARMOR returns only the root of the semantic graph, *i.e.* the situation could not be classified. Then the two reasoner exchange a synchronisation signal and, when each of those is aware that the other finished the classification phase continues their computation. In this case, ${}_{S}SR$ learns and structures a new item in the memory with an initial score and counters (at Line 14), while (at Line 15) the ${}_{E}SR$ only update the counters of all the j-th items given at Line 10. Remarkably, as discussed in Section 10.9, ${}_{E}$SR should also update the time instant associated with the classified items since this is required for computing the scores consistently. When ARMOR have processed the all requests (lines 16 and 18), each storing reasoner generate an internal stimulus to trigger the consolidation of the items (at lines 17 and 19).

10.10.4 Consolidating Reasoner

Each time the storing and the retrieving functions are computed from the relative reasoner, they trigger CR since their operation requires to recompute all the item's score. This is due to our definition of the functions to compute the scores (in equations 10.8 and 10.10), but other approaches should be developed keeping into account performances issues that recompute all the scores each time might arise for a real application.

In our scenario, we preliminary implemented a consolidating function based on a weighted sum of some components. Since our implementation requires to update all the scores at each new stored or retrieved operation, we always use the score computed during the previous situation and, we update the score for the current situation after any storing or retrieving operations. In accordance with equations 10.8 and 10.10, those components are such to increase the score of an item if (*i*) it is an item not stored for the first time or retrieved at the current instant of time, (*ii*) it models a complex situation, *i.e.* we award simpler items, (*iii*) it is a more general item (*i.e.* a parent node) of items with a high score, and (*iv*) only for ${}_{E}$CR, it is related to a recent instant of time.

In particular, when CR is triggered it query the memory graph through ARMOR (at lines 20 and 21 of Figure 10.7), which returns (at lines 22 and 23) all the items in the memory, included their relations, *e.g.* edges in the graph. In this way, CR receives the definitions of all the items, as well as their counters and scores, but it could also query further knowledge through ARMOR. Given this information, CR is

supposed to compute a new score for each item and saves all the new values in the ontology through ARMOR (at lines 24 and 25). At lines 26 and 27 ARMOR returns a consistency signals that in this paper is ignored and at lines 28 and 30 the forgetting reasoners are triggered.

10.10.5 Forgetting Reasoner

FR is triggered all the times the score is updated to ensure that the classification of items in their score level is up to date, but this assumption could be relaxed if we allow some latency in the memory management. As mentioned we consider three score levels, the forgetting reasoner deletes items with a weak score, and it identifies items with high or low scores. The latter are considered during retrieving since it would not return items with a low score, as discussed in Section 10.6.1. In this way, items that are not retrievable are still represented in the memory and, if the consolidating function is such to increase their scores enough, they might become items with a high score. In our scenario, we implement this classification based on thresholds, but other techniques could be more effective. Therefore, we consider this classification as a process not performed from the OWL reasoner that might limit the design of classifying techniques. Nevertheless, FR can exploit ARMOR also for this purpose.

In particular, between lines 30 and 33 of Figure 10.7, the forgetting reasoner query the memory to ARMOR that returns two graphs. At lines 34 and 35, FR uses ARMOR to create knowledge in the ontology such to classify each item in a score level and terminates its computations (at lines 38 and 39). Since FR do not trigger any further internal reasoners, CR receives a returning signal (at line 40 and 41). Then, the SR terminates its computation and returns a signal to ER (at line 42 and 43).

Remarkably, during this process, FR might access the same knowledge of CR through ARMOR, *i.e.* the entire memory. We explicitly divide those two components for modularity purposes and also because they might be used with different frequencies. Indeed the retrieving function returns only items with a high score. Therefore, weak items that would not be removed from the memory yet would not affect the responses of the robot. Nevertheless, in this case, a low item might become weak and then low again without being removed if FR is not triggered.

10.10.6 Retrieving Reasoner

RR is a service that is triggered when a dialogue ends (at line 50 of Figure 10.7), *i.e.* when some facts are sent from DM, which is implemented as a finite state machine. RR encodes facts similarly to EM, and then it performs a preliminary classification attempt through

ARMOR (at lines 51 and 52), similarly to SR and in accordance with Section 10.7.3. If among the items classified by SIT there is a Φ_j such that the similarity value ${}^{j}d_{\epsilon} = 1$, we assume that the user provides the robot with an exhaustive description of the situation to be retrieved. Otherwise, RR uses the learning and structuring operation of SIT to reason on a dummy item that after the computation, *i.e.* before to return a signal to DM, is removed.

In Figure 10.7 we show an example of the first case occurs in the episodic memory (at Line 54), while the second case for the semantic memory (at Line 53). Therefore, at Line 55 ${}_{S}RR$ uses ARMOR for learning and structuring a the dummy item $\Phi_{\mathcal{R}}$, as well as querying its arrangement in the memory graph. While, at Line 56, ${}_{E}RR$ increase the counter assigned to all each item given by AMOR at line 54 and removes all the facts and beliefs since its ontological operation is terminated. On the other hand, ARMOR responses to the query provided by ${}_{S}RR$ at Line 58 and, at Line 59, RR updates the counters and removes the knowledge of the current instant of time as above. Finally, at lines 61 and 62, the semantic and episodic graphs of retrieved items are sent to DM, which generates feedback to the user.

Similarly to PIT, the dialogue manager is in charge to divide episodic to semantic facts based on the interfaces used for each SIT instances. This because the knowledge should be sent to the respective components, which returns two different ordered set of items to be explained to the user. The components are such to return always paths in the memory graph that contains the root, a leaf, and the classified or dummy item. In our scenario, the robot prints on screen the retrieved graphs as well as the restrictions of its items, and explains the item with the highest score among the two graphs explicitly. Remarkably, the problem of choosing a single item among all the retrieved paths is a not trivial, and it is out of the scope of this paper. Nevertheless, we believe that our architecture is a suitable tool for also investigating this open issue as well.

In our implementation, the user is supposed to tell the robot a set of facts concerning symbols of the semantic interface associated with each instance of SIT. For instance, the user might ask the robot which was that situation where "a big cone was on the right-hand side of a red sphere". In this case, we generate facts within independent instances, *e.g.* `CONE`: γ_1, `BIG`: γ_1, `SPHERE`: γ_2, `RED`: γ_2, and $\texttt{right}(\gamma_1, \gamma_2)$. The user might add more characteristics of the situation to retrieve, *e.g.* he or she might also say that the item to retrieve should also have "a cone on the right-hand side of a green sphere". In our simple implementation, this would generate other two instances γ_3 and γ_4, that would be processed by the robot that would reply based on the discussion of Section 10.7.3. Again we do not consider in this paper the problem of implementing an efficient dialogue, but our architecture can be easily used for evaluating different heuristics.

10.11 Conlcusions

The paper presented an architecture for encoding, storing, retrieving consolidating and forgetting structured memory items. In particular, we represented each memory item as a node of an oriented graph that was generated from demonstrations based on the SIT algorithm, which implements one-shot structured learning based on logic reasoning. On the one hand, we extended SIT for computing the above functionalities. On the other, we deployed in our architecture more instances of SIT. This because we wanted to represent different types of memories, and we show the architecture for a scenario where a semantic and an episodic memory have been implemented.

We exploited the semantic interface provided by SIT to generalise our implementation. In particular, we presented a design pattern that does not limit the number of SIT instances provided with a different interface, and consequently the number of structures representing experiences with different semantics. Thanks to the purely symbolical method formalised in the SIT algorithm, we showed that an expressive set of memory models could be implemented in the architecture. This is aligned with our objective, which was to present a system that supports the evaluation of different implementations of memory functionalities.

Our architecture can represent knowledge in a contextualised and structured manner, where the semantics of the different structures can be customised through some parameters. For each of those structures, we proposed a set of extensible tools to perform all the memory functionalities named above. With investigating purposes, we gave particular support to the parameters that can be used for implementing consolidating and forgetting behaviours.

We presented a possible implementation of our architecture in ROS, where we used the same design pattern to model each functionality of the memory. Also, we took into account the issues to interfacing different perceiving and acting components. Through a centralised knowledge interfaced and services, we presented an architecture that requires as input triples involving symbols of instances of the environment. We focus on an architecture that is robust from the point of view of synchronisation among components, which allowed us to achieve a system flexible enough for evaluating several memory management implementations.

We presented a simplified tabletop scenario for which we proposed a complete implementation of the architecture with the purposes to highlight its features and limitations. For the referenced scenario, we showed how the robot could use an RGB-D camera for storing situations into memory items, with different semantics. Also, we showed how items could be associated with new situations and retrieved based on dialogues, also when concerning situation not directly experienced.

In particular, in our scenario, a user might ask to retrieve an item given a partial set of a characteristic describing a possible situation. The robot uses them to retrieve models of previous experiences that might be more or less complex than the one requested by the user. The items retrieved by the robot are ranked with a score, which is used with consolidating and forgetting purposes based on the structure of the memory itself. We consider in the semantic memory situation be described through abstract concepts as object shapes and qualitative spatial relationships. While, in the episodic memory we consider more concrete properties of objects, such as the colour and the dimension, as well as temporal and contextual information, *e.g.* the instant of acquisition and the particular table in which the objects were located. Those two different descriptions of the environment are represented with dedicated semantic interfaces used from two independent instances of SIT, which represent experiences in two graphs having different semantics. In our scenario, the user might interact with the robot using those symbols, which are directly mappable from sentences into the robot's memory, *e.g.* "a big, red, sphere that had a cone on the right-hand side".

From our preliminary test, we observed that the main limitation of the architecture is the computation complexity, especially for applications that require large knowledge representations. Nevertheless, in our design, we prefer to have a flexible than an optimised architecture. This because we would like to have an architecture for prototyping behaviours for managing memories. Unfortunately, as a drawback, we obtained a system that demands high computational resources. Once the behaviour has been tuned for an application through our architecture, we argue for more optimised implementations of such behaviour for a real deployment.

As a further work, we want to include the fuzzy implementation of the SIT algorithm and systematically test different consolidating and forgetting heuristics also based on more realistic scenarios. Also, we want to include mechanisms for automatically retrieve items based on visual perception, *e.g.* to recall an item which is similar to the current situation, with planning purposes for instance.

Part VI

Summary

This part concludes the thesis and discusses our approach.

11

Conclusions

In this thesis, we presented developing guidelines and tools for implementing architectures centred in a structured memory, which contains experiences that the robot can recall and share with persons. Through use cases concerning a tabletop scenario where objects are spatially arranged, we discussed its software components and their interfaces. Our architecture is designed to be used as a base for the iterative development process since it supports integrations and extensions for a broad range of scenarios. In particular, we consider scenarios where a user and a robot need to interact during a knowledge bootstrapping phase for sharing qualitative information that will be later used for achieving a common goal.

Our design pattern is based on the novel concept of semantic interface, which is used to interconnect the components of a ROS architecture with knowledge representations. We considered OWL ontologies provided with the Pellet and the fuzzyDL reasoners for representing knowledge in the robot memory, and we presented tools to design semantic interfaces, *i.e.* OWLOOP, in Chapter 5, and ARMOR, in Chapter 6. Such an interface allows components to rely on knowledge independently from other components that reason on the memory with aggregating purposes. In other words, through a semantic interface, we specify the syntax in which a component will store and retrieve knowledge in/from an ontology without defining its semantics, which would depend on the other modules of the architecture and prior knowledge.

Based on such a semantic interface, we presented in Section 1.4 an overview of the architecture which involves *external components* and *reasoners* relying on a *knowledge representation*. We test this design pattern in chapters 9 and 10. The former focussed on external components for engaging dialogues with a user that supervises robot beliefs, while the latter extended the system with the definition of different reasoners, which maintain a semantic and episodic memory through consolidating and forgetting mechanisms.

In the thesis, we based the reasoners of the architecture on the Scene Identification and Tagging algorithm (SIT), which is detailed in Chapter 7. We formalised the SIT algorithm to generate experience graphs based on demonstrations, and Chapter 8 proposes an extension

designed for uncertain environments. In particular, SIT takes symbolic facts describing a scene in an ontology as input, and it generates a definition of a category that will classify *similar* scenes when the robot will re-experience them. SIT exploits OWL reasoning to structure learned categories in an implication graph, which is given as output to represent a scene through classified sub-scenes previously experienced. Remarkably, SIT uses a purely logical representation, and it can evaluate the symbolic definition of qualitative scenes features for measuring their similarities.

The algorithm is designed with a particular semantic interface that describes the scenes through facts representing relations between salient elements of the environment. We coupled the algorithm with external components that require and provide knowledge from/to the ontology representing spatially arranged objects. In particular, we identified a semantic interface based on qualitative spatial relation, such "on the right-hand side of" or "behind", which are applied between objects with some characteristics, *e.g.* shape and colour. Based on this scenario, the thesis evaluates the maintenance of the knowledge in memory for some use cases, as well as the semantics of the memory graph that the robot structured over demonstrations.

For testing our design pattern, we presented in Chapter 2 the Primitive Identification and Tagging (PIT) architecture, which evaluates RGB-D data for creating instances in the ontology that describe objects with a position, a colour, a shape and some geometrical parameters, *e.g.* a radius if a sphere is segmented from the point cloud. Also, we presented in Chapter 4 an architecture that performs multimodal perception fusion for generating further descriptions of the scene in the ontology; for instance, we used it for fusing the description provided by PIT with the recognition obtained through convolutional neural networks.

Moreover, in Chapter 3 we present the Concept Action Grammar Generator (CAGG) for parsing sentences in a simplified English and obtained semantic tags related to words. The latter is used for engaging dialogues through the ROS architecture proposed in Chapter 9, where we focused on the interface between an ontology and an external component that both generates and consumes knowledge in memory. Remarkably, thanks to OWLOOP and ARMOR, we could define the semantic interface of a general component that has been used for access the ontology and implements four different types of dialogues.

In the architectures presented in Part v we consider the robot as a passive observer that can speak with the user and maintain its memory. Nevertheless, as presented in Chapter 6, ARMOR has been used to support symbolic adaptative task planners used for manipulation tasks. Also, Appendix A presents an architecture based on a fuzzy memory representation similar to the one used for formalising the fuzzy extension of SIT presented in Chapter 8). That paper shows how Monte Carlo simulation can be used for mapping symbolic spatial representation to a quantitative location that a robot should reach.

Also, it presents an anytime implementation of an architecture where multimodal interactions, including speech and pointing gestures, are based on a symbolic memory that drives the motion controller of a mobile robot.

11.1 Discussions

In Section 1.2, we introduce the requirements of the architecture concerning feasibility, shareability, expressivity, explicability, robustness and supports long-term applications. In the thesis, we discussed the feasibility of the architecture through some preliminary measurements showing that it can respect soft real time requirements for HRI. Also, we showed that the architecture is sharable among developers since it is based on mainstream and standard tools. Thanks to the definition of the semantic interface, the architecture can be used to represent expressive and structured memories, which contains knowledge that the robot can explain. Moreover, using a fuzzy ontology, the architecture can rely on a representation of uncertain environments, and this makes it robust to both perception noise and vague knowledge. Finally, we show an example of long-term memory management exploiting forgetting behaviour based on consolidating scores.

In Section 1.2 we also listed the features of the architecture we wanted to achieve and evaluate. In particular, we aimed to design a general architecture and, through semantic interfaces, we achieved such a feature for a large set of applications related to the requirements above. Similarly, we discussed and proposed examples for showing that the architecture is modular, and can be used for testing and evaluating different approaches for structuring experiences over time. Nevertheless, the scalability features of the architecture is a critical issue. Indeed, even if we tried to optimise the knowledge to be represented, the amount of resources needed for maintaining the memory scales exponentially. This is a current challenging issue, mainly due to OWL reasoning complexity.

We showed that the architecture well performs for scenarios where qualitative knowledge is required. Through different configurations of the semantic interface required by SIT, compared in Chapter 10, we presented how the memory can be structured based on different symbolic semantics, which highlights the flexibility of our design pattern. Since we used logic representation, we could build a general architecture because we consider abstract symbolic knowledge. Nevertheless, symbolic representation suffers the anchoring problem [42], and map sensory data into a symbolic description is not always trivial. Indeed, our memory is limited when the application requires quantitative reasoning, *e.g.* for physical interaction for instance. In this case, we argue for an architecture with symbolic, sub-symbolic, and numerical layers, similarly to other OWL-based architecture for robotic, *e.g.* [18, 29, 134].

In the thesis, we presented some implementation for acquiring sensory data and generate logic representations of the environment. Although this is a challenging open issue that must be tackled for real application, the thesis mainly focuses on representations of symbolic knowledge in memory, and not in the perception and actions based on such descriptions. Nevertheless, we wanted to deploy our architecture in a real environment to support the iterative development process, which relies on user's feedback from the early stage of the development, as presented in Chapter 1. Therefore, we implemented simple components that provide and requires knowledge in the memory with the purpose to bootstrap the development for the referenced spatial scenario and improve the architecture in the further iterations of the development process. Those components, presented in Part ii, might exploit more sophisticated state of the art approaches, which would certainly improve the overall performances of the architecture.

In our experiments, we involved few users, since our objective was to test the flexibility of the architecture before to evaluate the interaction. Already with these preliminary tests, we identify several heuristics for representing spatial relations, anchoring symbols to objects, as well as for consolidating and forget behaviours. Unfortunately, we did not found any evidence of the performances we would obtain with such heuristics, and we were unable to test them with a sufficiently high number of user for deducing their properties. Nonetheless, we experienced that our architecture can easily accommodate different approaches to be validated. Also, since we used mainstream packages, and thanks to a representation of knowledge that can be easily understood from a person, our architecture might also help in comparing results among different domain and developing groups.

11.2 Further Works

As a further work, we want to proceed with the next iterations of the development process and exploit the architecture for evaluating and comparing different heuristics for improving the interaction for knowledge bootstrapping in the referenced spatial scenario. Also, we want to use the same design guidelines for building an architecture that maintains a memory where experiences are structured based on the affordance of objects in the environment, as well as user's activities, their intentions and temporal reasoning.

Furthermore, the proposed architecture still needs a systematic evaluation of long-term applications, which will allow to better asses also scalability features. Moreover, planners and motion controllers should be further tested for making the robot acting and evaluate the behaviour of the architecture. This would enable to test more complex scenario where the interaction can be evaluated based not only on the knowledge bootstrapping phase.

Part VII

Appendix

This part introduces a preliminary implementation for acting based on a simple memory.

A

A Scalable Architecture to Design Multi-Modal Interactions for Qualitative Robot Navigation

Abstract

The paper discusses an approach for teleoperating a mobile robot based on qualitative spatial relations, which are instructed through speech-based and deictic commands. Given a workspace containing a robot, a user and some objects, we exploit fuzzy reasoning criteria to describe the pertinence map between the locations in the workspace and qualitative commands incrementally acquired. We discuss the modularity features of the used reasoning technique through some use cases addressing a conjunction of spatial kernels. In particular, we address the problem of finding a suitable target location from a set of qualitative spatial relations based on symbolic reasoning and Monte Carlo simulations. Our architecture is analyzed in a scenario considering simple kernels and an almost-perfect perception of the environment. Nevertheless, the presented approach is modular and scalable, and it could be also exploited to design application where multi-modal qualitative interactions are considered.

robot teleoperation · fuzzy spatial relations · multi-modal robot interaction

Authors

Luca Buoncompagni*, Suman Ghosh*,
Mateus Moura* and Fulvio Mastrogiovanni

Affiliation

University of Genoa.

*Published in the Proceedings of the 17th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018, Trento, IT).*

*These authors contributed equally to this work.

a.1 Introduction

Nowadays, robots are being deployed in daily-life environments with health-care and assistive purposes. Those applications usually exploit distributed environmental sensors, which provide information about a user's state and activities for experts that evaluate the well-being of a person. In some scenarios, caregivers might use telepresence robots to communicate with the person and ask for assessments that sensors do not provide. Those robots are required to interface smoothly with persons, especially elderly adults, taking into account social and spatial cognition. In other similar scenarios, pet-like robots, such as Miro [125] have been developed as companions for a long-term motivation of users with interaction and mobility difficulties. For both scenarios, the navigation of the robot should allow different degrees of autonomy to meet different users' needs [121]. In this paper, we address the issue of locating a mobile robot with a high degree of autonomy, controlled through qualitative incremental commands.

In these scenarios, robots should be able to navigate in an environment shared with humans, who might like to naturally issue commands that drive the robot to specific areas of the room, with no high precision requirements. For instance, if a child stays in the bedroom for a long time during a sunny day playing with his or her pet-like robot, the mother could ask the robot to go *close to the exit door*, trying to motivate the child to play in the garden. On the other hand, if an older adult is on the sofa and receives a call from the doctor; he or she might ask a telepresence robot to locate itself in a comfortable position, for instance, *in front of the television and on the right-hand side of the table*. Generally, we consider the teleoperation based on qualitative commands all the times that a user cannot give either geometrical or velocity references.

Robot teleoperation is a largely investigated problem in the literature, especially for exploration and manipulation tasks, where a certain precision is required. Several types of systems exploit different interfacing devices such as joysticks, smartphones, Leap Motions, smartwatches, eye-tracking technology, etc. Different applications lead to different approaches; for instance, a reacting control is suitable for low-level interacting behaviors. Instead, a high-level interaction usually requires that the user provides additionally informations; typically, velocity or geometrical references, *e.g. go slower*, *go forward*, etc. Remarkably, while the user is issuing those commands, he or she is required to continuously operate the robot, since its behavior also depends on the states it had at previous instances of time. In other words, the robot has a memory, and the users operate it by giving differential references, which could be based on previous positions or speeds.

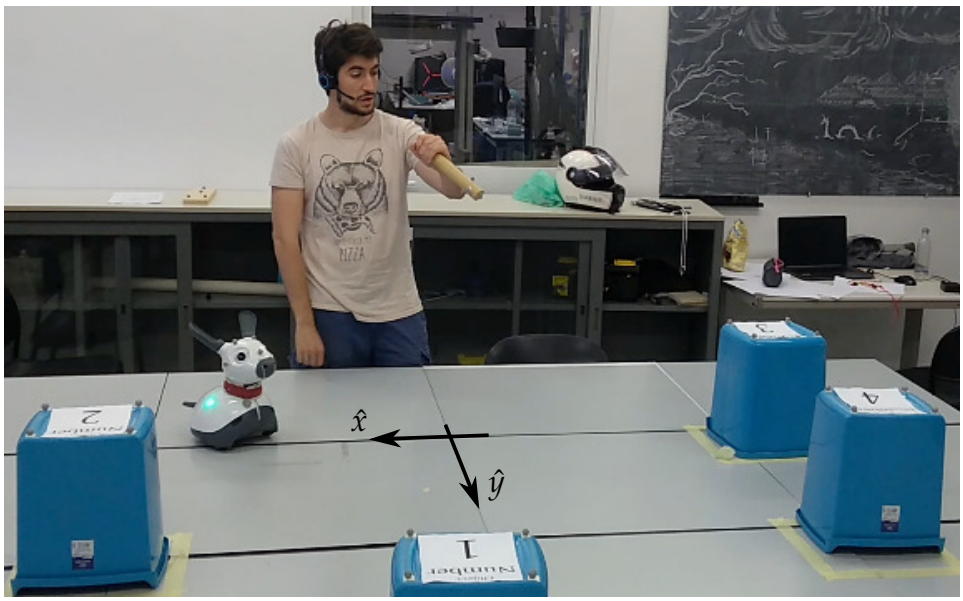


Figure A.1: The referencing scenario.

We designed an interaction pattern that relies on a robot having in memory a set of commands that restrict the possible target position in space. These restrictions are described qualitatively concerning the robot and generic objects, seen from the user's viewpoint. The perception of such knowledge from visual and auditory channels is an extremely challenging problem, especially for general-purpose systems. Due to a large number of uncontrolled variables influencing the experiments, human-robot interaction applications are typically designed through the *iterative developing* process based on the (*i*) developing, (*ii*) testing, and (*iii*) evaluating phases, which are performed in an iterative manner. During *ii* and *iii*, some measurement of the interaction, as used in [116], can guide further developments to be undertaken during the *i* phase of the next iteration. Remarkably, to do not bias experiments outcomes with developers beliefs, it is suggested that the *i* phase tries to improve the system with minor changes, *i.e.* the evaluation frequency should be high.

If verbal commands are used to guide the robot to a target position, symbolic representations are required to understand the user's sentences and find a target pose suitable for the given commands. Typically, in such cases, systems exploit a map between geometrical positions (or velocities) and symbols, for each known location, based on the user's inputs [120, 130]. During verbal interaction, users also communicate through other modalities, such as deictic gestures. In many works multi-modal robot interaction is investigated – for instance, [43] proposes to use Reinforcement Learning and applied associative memories – for fusing audio and gestures, and instruct the robot on qualitative commands for picking and place tasks. For merging such interaction modalities, a Markov Decision Process in [101],

Recurrent Neural Networks in [7], and a Dynamic Bayesian Network in [76], have been used. Remarkably, not only do such methods allow dealing with uncertainties, but they can also be used for *learning* behaviors in a data-driven manner. However, the models generated by these techniques are typically not modular, since the introduction of a new behavior might affect the probability distribution of all the other states). Thus, the system can not always scale among iterative development steps.

Fuzzy logic formalism, mainly developed to be used with natural language, can be used for representing the robot position with respect to shape, position and orientation of general objects. In particular, [154] proposes a mapping function between sentences (containing spatial relations) and a fuzzy membership value associated with a physical location, while [149] uses fuzzy membership functions for fusing gestures and natural language. Moreover, [20] proposes mathematical morphology for describing north, south, east and west points of objects or rooms. Fuzzy morphology has also been developed for image processing. [48] presents a kernel-based approach for classifying spatial relationships between hand-drawn symbols, where such kernels are generated through a data-driven approach based on Support Vector Machines.

In this work, we developed a system that allows symbolic teleoperation of a mobile robot based on fuzzy spatial relations between objects issued through multi-modal commands. Our main contribution is a software architecture defining the required components and interfaces to be used during the steps undertaken in an iterative development process. Our objective is to start designing and testing a modular and scalable reasoning technique that can be used in dynamic environments and with a broad set of qualitative commands, possibly customized for each user and scenario. We present our architecture through a preliminary implementation suitable for the *i* phase of the first iteration (*i.e.* concerning a simplified, but real use case shown in Figure A.1), and we discuss its scalability and modality features achieved through fuzzy reasoning and Monte Carlo simulations.

In particular, we address the problem of representing qualitative spatial references and finding a suitable target position that the robot should reach. In this scenario, we assumed a robot that perfectly knows its location, the locations of referenced objects and their *identifiers*. Also, it can perfectly translate the user's voice in text, and detect his or her pointing gestures. Moreover, in the presented use case, the user interacts only though combinations of five spatial relations (*i.e.* in *front* of, *behind* of, on the *right* of, on the *left* of, and *near* to). Nevertheless, the paper shows how the architecture can accommodate and merge different spatial relation models, based also on some different input modalities, in a modular manner.

Section A.2 gives an overview of our system, while in Section A.3 we introduce the user interfacing pattern for the referenced use case. In Section A.4 we present a fuzzy spatial reasoner based on fuzzy kernels, which are further detailed in Section A.5. Section A.6 presents our experimental evaluation and discusses preliminary results. Conclusion follows in Section A.7.

a.2 Software Architecture

Figure A.2 presents the proposed interaction patterns from a Component-Based Software Engineering (CBSE) perspective, graphically shown through a structural Unified Modeling Language (UML) diagram. In the Figure, boxes represent software *components*, which provide and require *interfaces* to operate. The latter are graphically identified with the ball-socket notation. All interfaces are described by a specific *message* defining the data type exchanged among components through *ports* (*i.e.* data communication channels). We specify the proposed components for the widely adopted Robot Operating System (ROS) framework. Therefore, we assume a component to be a *node* or a *service*, the former characterized by a *spinning rate*, while the latter provides services on demand.

Within an area monitored by a motion capture system, the Rigid Body Detector component provides to the robot the position of all the elements of our referencing scenario. The robot process inputs coming from the Speech Processor component, that parses sentences to providing semantic tags for specific words, which are used for identifying qualitative regions in space that contain the target position.

When the user says a sentence containing a valid command (*i.e.* meaningful semantic tags are found in the text translation of the person's voice), and eventually performs a deictic gesture, the relative inputs are provided to the Command Logic component, implemented as a finite state machine. In particular, the robot is initially in a (*i*) *ready* state and when a command is issued, to the Command Logic goes to a (*ii*) *listening* state. It can lead to an (*iii*) *invalid* state if the command is not understood, or to a (*iv*) *computing* state, which holds until a suitable target position is found. When this occurs, the robot starts moving and returns to the ready state; otherwise an (*v*) *inconsistent* state emerges, and the spatial beliefs of the robot are erased.

The Command Logic maintains in the memory the user's instructions that the robot should take into account for finding a suitable target point. Such a memory is implemented as a buffer that contains all the vocal and gesture commands of the user until either the target position is found, or an inconsistent state is reached, or the user explicitly asks for a new interaction pattern. In all those cases the buffer is cleaned and the interaction restarts.

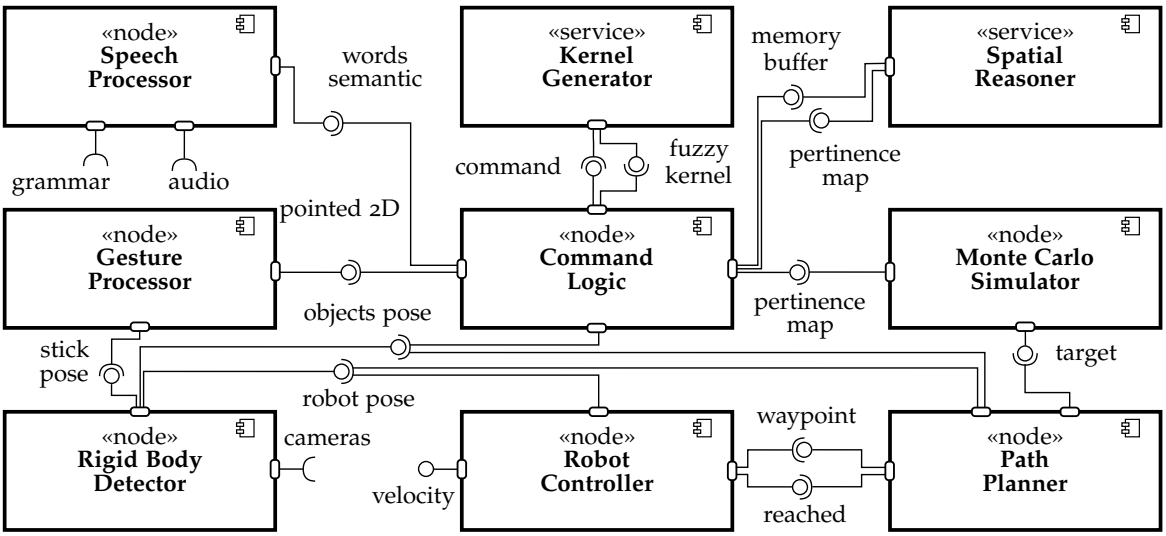


Figure A.2: The ROS-based software architecture.

Each element of the memory buffer is a representation of the suitable positions that respect a single command, *i.e.* a fuzzy kernel, provided by the Kernel Generator component. The Spatial Reasoner component merges all representations in the buffer to find an overall fuzzy landscape, which describes the degree of pertinence of each location in the workspace with respect to the given commands. Since we design the system for accommodating commands based on generic spatial relations, we consider the fuzzy pertinence map to be a general function spanning in values between $[0, 1]$, where the higher the pertinence value represents the most suitable target candidate point.

The Monte Carlo Simulator component solves the problem of findings a suitable target in such a generic landscape. It evaluates the pertinence of randomized locations until certain conditions are met. As heuristic conditions, we posed (*i*) a very high threshold after which the simulations are terminated, (*ii*) upper and lower pertinence bounds for which the target position is generated, although the simulations continue looking for improvements, and (*iii*) a limit on the maximum number of simulations. When the Monte Carlo Simulator produces a target point with fuzzy pertinence value higher than in the previous simulation, the new target is provided to the Path Planner component, implemented using the Rapidly-Exploring Random Tree (RTT*) algorithm, which consequentially provides waypoints, used as references for the low-level Robot Controller.

This architecture allows a continuous interaction, since new commands are stored in a buffer and are evaluated as soon as previous computations ends, *i.e.* as soon as Monte Carlo simulation provides a suitable point. Therefore, while the robot is moving towards a target, the user can guide it reactively. In this paper, we evaluate such property through the estimation of the time spent in the computing state,

especially as far as the complexity of the Kernel Generator and the Spatial Reasoner are concerned. However, it is important to highlight that for a real application the complexity of finding a path and perceiving the environment might introduce not negligible delays before the robot starts moving, and consequently affect user's reaction time.

a.3 Multi-Modal Interface

In this section we show how a user can interface with the robot in our simplified scenario. In particular, he or she can instruct the robot using five types of spatial relations, each associated with a specific fuzzy kernel. We design an interaction pattern where the user mainly uses sentences to guide the robot. However, he can further enforce and enhance his command by pointing to a specific location.

We suppose that the commands are always between the robot and a well-defined object, *e.g. go in front of box number 1*. For understanding these types of sentences, we used the Concept-Action Grammar Generator (CAGG) presented in Chapter 3 which, given a text resulting from the speech-to-text process, returns the semantics of the sentence through tags associated to specific keywords. Such association is based on rules encoded on a Backus-Naur Form (BNF) grammar of the form:

```
<go> : <relation> <object>,                    (A.1)
```

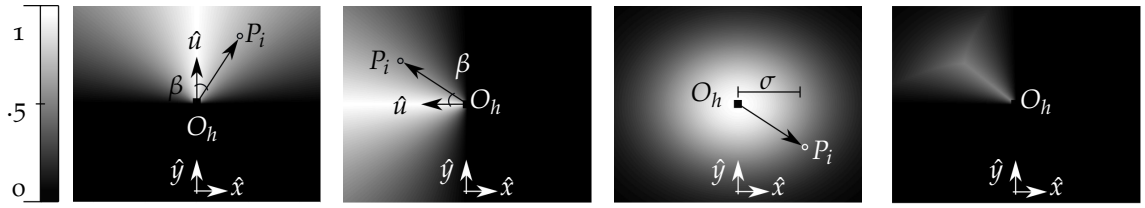
where each identifier between angled brackets is associated to a specific set of keywords to be matched in a sentence for obtaining the related semantic tag (*e.g.* `[front, 1]`).

Moreover, through simple sentences is also possible to give other directives to the robot, such as to mark the task *done* (when the interaction is complete), and to *reset* the buffer (to clean the commands memory). These types of commands are recognized through a different grammar that allows changes of the Command Logic state, as detailed:

```
<directive> : <go> | <reset> | <done>.         (A.2)
```

Therefore, for recognizing a user sentence we use a hierarchy of grammars, where Rule A.2 is on the top layer and Rule A.1 is evaluated only if the `go` directive is found.

Last but not the least, the motion caption system tracks the orientation of a stick that the user holds in his hand (Figure A.1) while performing deictic gestures. When the user performs a pointing gesture, we compute the intersection point of the line generated by the stick orientation and the ground plane containing the robot and objects. The robot is instructed to be *near* a point computed as above. It is to be noted that pointing is typically a not precise source of information, especially when the user is standing at a distance from the scene. We



(b) $\mathcal{K}_1$: `behind` O_h. (c) $\mathcal{K}_2$: `left` O_h. (d) $\mathcal{K}_3$: `near` O_h. (e) $\mathcal{M}{=}\mathcal{K}_1{\otimes}\mathcal{K}_2{\otimes}\mathcal{K}_3$.

Figure A.3: Three fuzzy spatial kernels applied to the same object O_h (A.3c, A.3b, and A.3d) and their conjunction, *i.e.* the pertinence map (A.3e).

thus consider the identified point in the ground to be qualitative as an explicit spatial relation given through a spoken sentence (*e.g. go near box 2*). In other words, the *near* spatial relation generated by a pointing gesture accounts for non-negligible pointing errors by the user.

a.4 Spatial Reasoner

Within a workspace $W \subseteq \mathbb{R}^2$, we assign the robot goal position by maximizing the qualitative fuzzy values represented in the landscape of possible target positions. For our use case, we divided the workspace into discretized rectangles of equal size. For each unit rectangular element, we select the center point P as its location. Thus, we have a set of possible locations $\{P_i,\ \forall i \in [1,n]\} \subset W$. In this scenario, the fuzzy landscape $\mathcal{M} \subset \mathbb{R}^3$ is a pertinence map that assigns to each point in the workspace a fuzzy membership $\mathcal{M}(P_i) = \mu_i \in [0,1]$, which represents how pertinent a location is with respect to all the spatial restrictions contained in the robot's memory. As we mentioned before, for each command issued by the user, the system computes a fuzzy spatial kernel $\mathcal{K} \subset \mathbb{R}^3$ which represents the fuzzy membership of each point of the workspace for just for a single command. Therefore, a generic kernel $\mathcal{K}_j$ embeds for each point of the workspace, a fuzzy membership value $\mathcal{K}_j(P_i) = \mu_{ji} \in [0,1]$, that describes its pertinence for a single spatial relation, as shown in Figure A.3.

Given m commands in the robot memory, a set of $B = \{\mathcal{K}_j,\ \forall j \in [1,m]\}$ kernels are generated. The system then computes an overall pertinence map through fuzzy conjunctions considering the minimum value of the fuzzy membership among all kernels in B. This is done for each point P_i of the workspace. Formally,

$$\mathcal{M}(P_i) = \bigotimes_{j=1}^{m} \mathcal{K}_j(P_i) = \min_{j \in B} \left(\mu_{ji}\right), \tag{A.3}$$

represents the fuzzy pertinence of the i-th point to satisfy all the m commands.

For example, Figure A.3e shows the intersection between relations ($\mathcal{K}_1$) `left`, ($\mathcal{K}_2$) `behind`, and ($\mathcal{K}_3$) `near`, applied to the same object O_h located in the center of the workspace. Brighter regions of the workspace indicate consistent locations of a target point that respects all three given commands. Remarkably, with this formalization, it is possible to see kernels as generic modules that represent the intersection of different spatial relations, applied in different contexts. From a general perspective, all kernels can (*i*) represent fuzzy membership values between 0 and 1, (*ii*) for each point of the workspace.

If we consider a robot that evaluates the pertinence map $\mathcal{M}^t(P_i)$ (containing fuzzy membership values μ_i^t) for each time instance t when a new command is issued, the computation of Equation A.3 can be simplified and evaluated as:

$$\mathcal{M}^{t+1}(P_i) = \mathcal{M}^t(P_i) \otimes \mathcal{K}_{m+1}(P_i) = \min\left(\mu_i^t,\ \mu_{m+1,i}\right), \tag{A.4}$$

which denotes a computation complexity of $\mathcal{O}(n)$ since it requires an element-to-element comparison between two sets of size n. This assures reasoning scalability because Equation A.4 computes the pertinence map in an incremental manner and its computation cost does not increase with the size of commands in the robot memory, but always remains constant if the workspace is fixed. Nevertheless, with this approach, we lose information concerning previous instances in time. Hence, backtracking to previous valid pertinence landscape (*i.e.* removing the effects of $\mathcal{K}_j$ from $\mathcal{M}^t$) remains an open issue.

In the pertinence landscape $\mathcal{M}$, we can define a target position $T*$ as a location with high fuzzy value, formally:

$$T^* = \underset{P_i}{\operatorname{argmax}}\, \mathcal{M}(P_i) \equiv \{P_i : \mathcal{M}(P_i) = \mu_i \to 1\} \subset W. \tag{A.5}$$

Unfortunately, for generic kernels, this problem is not well-posed since there might exist many points with the same high pertinence value. Indeed, even if in the referenced scenario we used simple kernels based on heuristics (discussed in the next section), Figure A.3e shows more than one suitable target point with sufficiently high fuzzy pertinence.

To address this issue and choose a quantitative target position T with a fuzzy pertinence similar to T^*, we used the randomized Monte Carlo simulation approach introduced in Section A.2. At the end of each q-th simulation, new randomized locations P_i are evaluated. Among them, the point with the best pertinence is elected the new target T^q, *iff* it represents a more pertinent solution than the one selected during the previous simulation, *i.e.* $\mathcal{M}(q^t) > \mathcal{M}(T^{q-1})$.

a.5 Spatial Kernel

Figures A.3c and A.3b show the kernels used for commands `left` and `behind` respectively. We adopt a simplification of the formalization proposed in [19], which defines fuzzy pertinence of spatial relations between objects with a general shape. In this work, we reason on the spatial relations in a scene having w point-like objects O_h, *i.e.* a set of occupied locations $\{O_h,\ \forall h \in [1, w]\} \subset W$. Through a command that satisfies Rule A.1, the user specifies an `object` O_h and, if specific *relation* are used (*i.e. left*, *behind*, *right*, and *front*), he or she specifies also an orientation $\hat{u}$. Thus, we posed

$$\mathcal{K}_j(P_i) = \max\left(0, 1 - \frac{2}{\pi}\beta\right) \quad \text{where} \quad \beta = \arccos \frac{\overrightarrow{O_h P_i} \cdot \hat{u}}{||\overrightarrow{O_h P_i}||}, \tag{A.6}$$

where $\hat{u}$ represents a unit vector along direction of kernel orientation with respect to a global frame $\hat{x}\hat{y}$ fixed as in Figure A.1. Throughout the paper, our global $\hat{y}$ axis is always directed from south to north in the images of fuzzy landscapes.

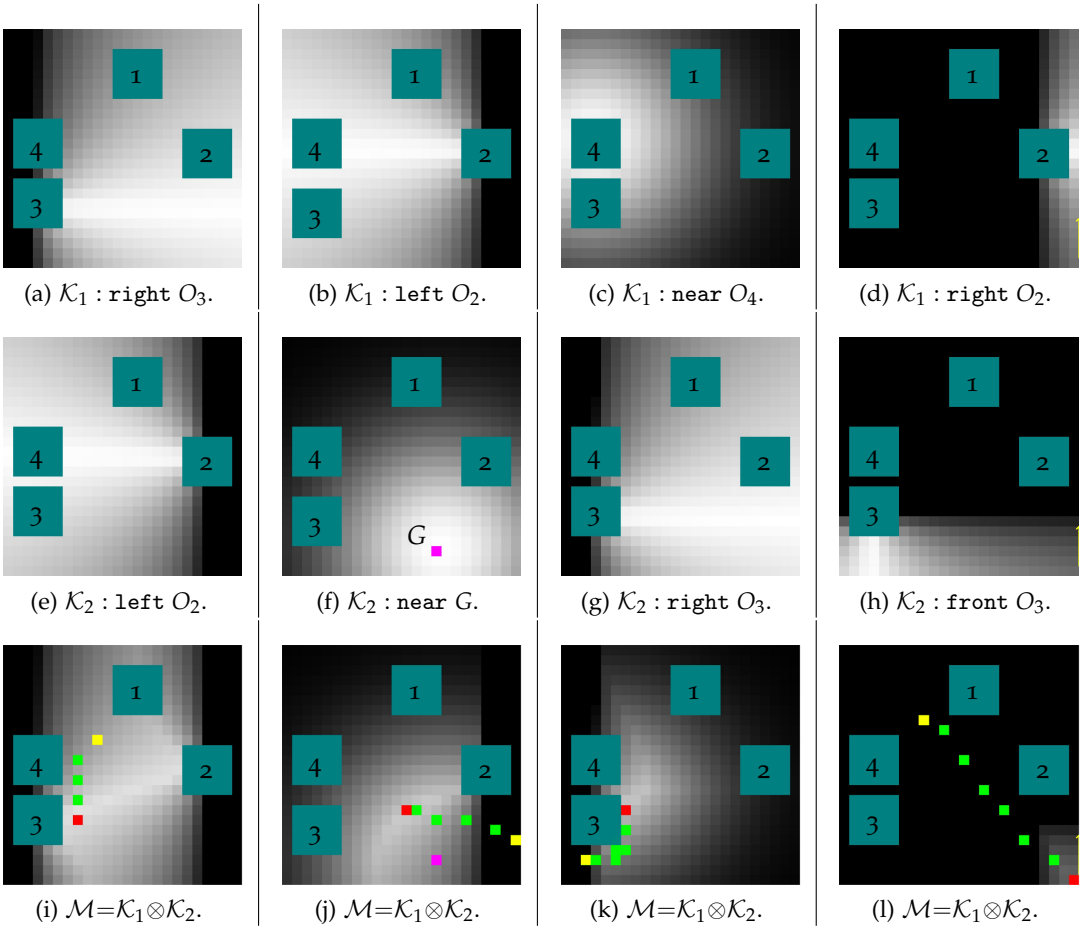
The user orientation is fixed along the $\hat{y}$ axis. The $\hat{u}$ direction is initially fixed along the $\hat{x}$ axis. Figure A.3b shows the kernel representing the `behind` relation, which is computed for $\hat{u}$ rotated anticlockwise by $\frac{\pi}{2}$ with respect to the $\hat{x}$ axis. Figure A.3c shows the `left` relation, computed for $\hat{u}$ rotated by π. Consequently, inverse relations such as `front` and `right` are computed for $\hat{u}$ rotated by $\frac{2}{3}\pi$ and 0 respectively. Remarkably, while we consider spatial relations between point-like objects, we also describe their encompassing bounding box, and force all the kernels to have $\mu_{ji} = 0$ for all P_i points within the w boxes.

The above kernel formulation cannot be used for representing a relation such as `near`, since Equation A.6 does not take into account the distances between P_i and O_h, but only the orientation of the connecting vector $\overrightarrow{O_h P_i}$. For representing a spatial pertinence related to the distance between the object O_h and a possible target point P_i, we used a Gaussian kernel (Figure A.3d) defined as

$$\mathcal{K}_j(P_i) = e^{-\frac{A^2}{2\sigma^2}} \quad \text{where} \quad A = ||\overrightarrow{O_h P_i}||, \tag{A.7}$$

where σ has been set as the width of the workspace divided by four; if W is considered to represent a square space, $\sigma = \frac{\sqrt{n}}{4}$.

Notably, the `near` restriction is the only relation of the referenced use case associated with the pointing gesture. In particular, if the user pointed to a specific location $G \subset W$ in the ground, we merely apply the kernel in Equation A.7, centered in G instead of the center of an object, *i.e.* $O_h \equiv G$.



(a) $\mathcal{K}_1$: `right` O_3. (b) $\mathcal{K}_1$: `left` O_2. (c) $\mathcal{K}_1$: `near` O_4. (d) $\mathcal{K}_1$: `right` O_2.

(e) $\mathcal{K}_2$: `left` O_2. (f) $\mathcal{K}_2$: `near` G. (g) $\mathcal{K}_2$: `right` O_3. (h) $\mathcal{K}_2$: `front` O_3.

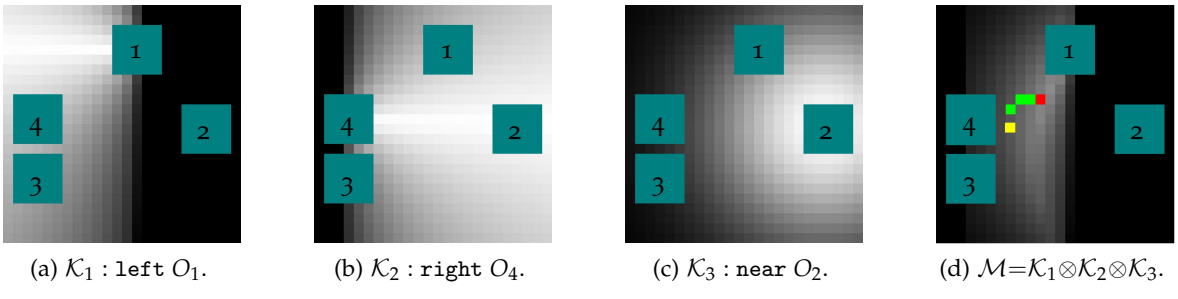
(i) $\mathcal{M}{=}\mathcal{K}_1{\otimes}\mathcal{K}_2$. (j) $\mathcal{M}{=}\mathcal{K}_1{\otimes}\mathcal{K}_2$. (k) $\mathcal{M}{=}\mathcal{K}_1{\otimes}\mathcal{K}_2$. (l) $\mathcal{M}{=}\mathcal{K}_1{\otimes}\mathcal{K}_2$.

Figure A.4: Four different instructions based on two kernels (shown by columns), centered in an object (enumerated boxes) or in a pointed location (purple cells). The third row shows the initial and target position (yellow and red cells respectively), and the followed path (green cells).

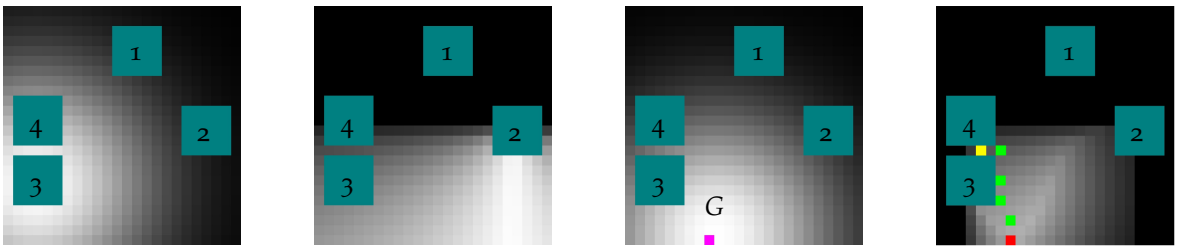
a.6 Use Case

Figure A.1 shows the scenario in which we test the system using a Miro robot [125] that navigates in an squared workspace of 5.76m² discretized into $n = 576$ unit blocks of size 0.01m². Within the workspace, four objects (identified through an enumeration) are statically located in such a way to have the center in a discretized position (*i.e.* $O_h \equiv P_j \ \forall h \in [1, w]$), and to occupy a given number of cells (figures A.4 and A.5). A user looks at the scene and instructs the robot using a microphone and a pointer. The global reference $\hat{xy}$ is fixed from the user's orientation, as shown in the figure.

We design an experimental script divided into three parts. Initially, we explain to the user the types of sentences understood by the robot, and how to perform valid pointing gestures. During the second phase, the user decides a target point T^* (that we record) and begins instructing the robot to try to reach that position. The test is performed again with a different target point before moving on to the third phase,



(a) $\mathcal{K}_1$: `left` O_1. (b) $\mathcal{K}_2$: `right` O_4. (c) $\mathcal{K}_3$: `near` O_2. (d) $\mathcal{M}{=}\mathcal{K}_1{\otimes}\mathcal{K}_2{\otimes}\mathcal{K}_3$.



(e) $\mathcal{K}_1$: `near` O_3. (f) $\mathcal{K}_2$: `front` O_2. (g) $\mathcal{K}_3$: `near` G. (h) $\mathcal{M}{=}\mathcal{K}_1{\otimes}\mathcal{K}_2{\otimes}\mathcal{K}_3$.

Figure A.5: Two instructions given through three commands (shown by rows), where the last column shows the robot navigation as in Figure A.4.

which is a repetition of the second phase, with the difference being that T^* is imposed on the user by us. The navigation ends when the user believes that the robot's target position T is qualitatively close enough to the intended location T^*; in this case, he is supposed to issue a `done` command (Rule A.2). The whole experiment took about half an hour. Here, we present some preliminary results collected with five volunteers.

Figures A.4 and A.5 show some interesting behaviors of the robot when the memory buffer contained two and three commands respectively. Both figures show the kernel independently, as well as their conjunction (*i.e.* the pertinence map), in which we superimpose the path traversed by the robot. Particularly, Figure A.4i shows the robot behavior in a case where the user said "Miro, go on the right of box 3" (Figure A.4a) "and on the left of box 2 please" (Figure A.4e). In the second column, Figure A.4j shows the robot's final location when the user said "go on the left of box 2" and pointed to G, while Figure A.4k shows a solution found if the robot is constrained to be on the right of O_3 and close to O_4. Figure A.4l illustrates a case where the conjunction of the user's instructions was almost inconsistent. Similarly, Figure A.5d demonstrates a case where the user later decided to provide a relation concerning object O_2 to move the robot more towards it, while remaining on the left of O_1, and on the right-hand side of O_4. Finally, Figure A.5h presents a scene where a pointing gesture is used to correct an erroneous previous instruction (`near` O_3) given by the user which caused the robot to move to the north of O_3, while she

actually intended to move the robot to the south of O_3 (*i.e.* *closer* from her viewpoint).

During all the experiments (performed with a processor Intel® Core™ *i*5-460*M* 2.53*GHz* and 4*GB* of memory), we compute Equation A.4 137 times with a buffer size spanning in $m \in [1, 9]$. For each times we compute the pertinence map the Spatial Reasoner spent 6 ± 2 ms, while the Monte Carlo simulator (which had an upper pertinence threshold of 0.95) spent 4 ± 2 ms for identifying a sufficiently high value. The generation of the navigation path is one of the most expensive tasks, which lasts for 4789 ± 1958 ms. In accordance with Equation A.4, the computation where no affected by the number of commands and scales linearly with the number of point in the workspace.

During the third phase of our experiments, we propose to the user some target T^* complex enough to be described given the five supported relations and the objects' locations (such as the one perfectly reached in Figure A.5h). We observed that the user spent time thinking about the correct command to issue (in average 40 ± 36 seconds). During all the experiments the robot memory was reset 34 times, in average every 4 commands, and we computed $\mathcal{M}$ with m=1 for 54 times, with m equal to 2, 3, 4 for 39, 24, 12 times respectively, and with $m \in [5, 9]$ less than 3 times. For all those maps, before to meet one of its terminating condition, the Monte Carlo simulations retrieved a fuzzy pertinence value of 0.82 ± 0.17. Particularly for the target points that the users tagged with a `done` command (T), we compute a distance of 0.080 ± 0.034 meters between that point and the initially intended target position T^*; reached with a fuzzy pertinence found by Monte Carlo of 0.78 ± 0.20.

a.7 Conclusions

We presented a preliminary evaluation of a mobile robot that can locate itself based on qualitative conjunctions of spatial relations concerning specific objects, issued through speech-based and deictic commands. Our architecture relies on fuzzy kernels that are used to represent a landscape indicating the pertinence of each point of the workspace to satisfy a set of the qualitative commands expressed by a user. We computed the robot position based on combinations of kernels and Monte Carlo simulations, and we observed a real robot navigate toward qualitative targets within a simplified scenario.

We described the scalability feature of our ROS-based architecture[1.1], as well as the modularity of its reasoning approach. We discussed the types of kernels that can be adopted and evaluated for more complex interaction semantics and modalities. In particular, we described its components and interfaces with the aim of reusing them during fur-

1.1 https://github.com/EmaroLab/mmodal_teleop

ther iterative development steps, toward a natural interaction in real applications.

The current implementation is limited due to the fact that the representation of the pointing gestures, the verbal commands and the objects orientation might depend on the user interpretation. Moreover, similarly to navigation algorithms based on potential fields, our approach relies on a one-to-one spatial mapping between a fuzzy representation and the workspace. Therefore, the inclusion of information which is not directly mappable to spatial semantics (*e.g.* voice volume) is an open issue in the reasoning process. Nevertheless, other interaction modalities (*e.g.* based on gaze) can be supported by the adopted fuzzy reasoning technique.

Since the number of users is not adequate for evaluating the naturalness of the interactions, we are currently collecting further data through more structured experiments. From a preliminary interaction assessment, we noted that the presented kernels are not always exhaustive, especially if no other spatial relations are included in the reasoning process, *e.g.* *between*, *far*, etc. Our future objective is to design additional kernels and perception modules to be evaluated through experiments involving the presented architecture.

References

[1] Lisi Francesca A and Straccia Umberto. "A Logic-Based Computational Method for the Automated Induction of Fuzzy Ontology Axioms." en. In: *Fundamenta Informaticae* 4 (2013), pp. 503–519. ISSN: 0169-2968. DOI: 10.3233/FI-2013-846 (cit. on p. 114).

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: a system for large-scale machine learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283 (cit. on p. 35).

[3] Marina Alberti, John Folkesson, and Patric Jensfelt. "Relational Approaches for Joint Object Classification and Scene Similarity Measurement in Indoor Environments." en. In: *2014 AAAI Spring Symposium Series*. Mar. 2014 (cit. on p. 68).

[4] Aitor Aldoma, Federico Tombari, Johann Prankl, Andreas Richtsfeld, Luigi Di Stefano, and Markus Vincze. "Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Karlsruhe, Germany, 2013, pp. 2104–2111 (cit. on p. 32).

[5] JAMES F. Allen. "Maintaining Knowledge about Temporal Intervals." In: *Readings in Qualitative Reasoning About Physical Systems*. Ed. by Daniel S. Weld and Johan de Kleer. Morgan Kaufmann, Jan. 1990, pp. 361–372. ISBN: 978-1-4832-1447-4. DOI: 10.1016/B978-1-4832-1447-4.50033-X (cit. on pp. 58, 68).

[6] John R. Anderson and Christian J. Lebiere. *The Atomic Components of Thought*. en. Psychology Press, Jan. 2014. ISBN: 978-1-317-77830-1 (cit. on p. 172).

[7] Alexandre Antunes, Gabriella Pizzuto, and Angelo Cangelosi. "Communication with Speech and Gestures: Applications of Recurrent Neural Networks to Robot Language Learning." In: *Proc. GLU International Workshop on Grounding Language Understanding*. 2017, pp. 4–7 (cit. on p. 218).

[8] R. C. Atkinson and R. M. Shiffrin. "Human Memory: A Proposed System and Its Control Processes." In: *Psychology of Learning and Motivation*. Ed. by Kenneth W. Spence and Janet Taylor Spence. Vol. 2. Academic Press, Jan. 1968, pp. 89–195. DOI: 10.1016/S0079-7421(08)60422-3 (cit. on p. 169).

[9] Pradeep K Atrey, M. Anwar Hossain, Abdulmotaleb El Saddik, and Mohan S. Kankanhalli. "Multimodal fusion for multimedia analysis: A survey." In: *Multimedia Systems* 16.6 (2010). DOI: 10.1007/s00530-010-0182-0 (cit. on p. 32).

[10] F. Baader, D. Galvanise, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic handbook: theory, implementation, and applications*. Cambridge, MA: Cambridge University Press, 2007 (cit. on p. 18).

[11] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *Introduction to Description Logic*. en. Cambridge University Press, Apr. 2017. ISBN: 978-0-521-87361-1 (cit. on pp. 4, 40, 58, 67, 110, 169).

[12] Marian Babik and Ladislav Hluchy. "Deep Integration of Python with Web Ontology Language." In: *Proc. of 2nd Workshop on Scripting for the Semantic Web*. 2006/June 12, 2006// (cit. on p. 41).

[13] Alan Baddeley. "The Episodic Buffer: A New Component of Working Memory?" English. In: *Trends in Cognitive Sciences* 4.11 (Nov. 2000), pp. 417–423. ISSN: 1364-6613, 1879-307X. DOI: 10.1016/S1364-6613(00)01538-2 (cit. on p. 170).

[14] F. Balint-Benczédi, Z. Márton, M. Durner, and M. Beetz. "Storing and Retrieving Perceptual Episodic Memories for Long-Term Manipulation Tasks." In: *2017 18th International Conference on Advanced Robotics (ICAR)*. July 2017, pp. 25–31. DOI: 10.1109/ICAR.2017.8023492 (cit. on p. 169).

[15] Liam J. Bannon. "Forgetting as a Feature, Not a Bug: The Dualityof Memory and Implications for Ubiquitous Computing." In: *CoDesign* 2.1 (Mar. 2006), pp. 3–15. ISSN: 1571-0882. DOI: 10.1080/15710880600608230 (cit. on p. 171).

[16] Pierre Barrouillet, Sophie Bernardin, Sophie Portrat, Evie Vergauwe, and Valérie Camos. "Time and Cognitive Load in Working Memory." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 33.3 (2007), pp. 570–585. ISSN: 1939-1285(Electronic),0278-7393(Print). DOI: 10.1037/0278-7393.33.3.570 (cit. on p. 169).

[17] Selena Sohaila Baset and Kilian Stoffel. "Object-Oriented Software Modeling with Ontologies Around - A Survey of Existing Approaches." en. In: *The 30th International Conference on Software Engineering and Knowledge Engineering*. July 2018, pp. 29–45. DOI: 10.18293/SEKE2018-198 (cit. on pp. 7, 40, 41).

[18] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels. "Know Rob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 512–519. DOI: 10.1109/ICRA.2018.8460964 (cit. on pp. 4, 40, 211).

[19] I. Bloch. "Fuzzy Relative Position between Objects in Image Processing: A Morphological Approach." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.7 (July 1999), pp. 657–664. ISSN: 0162-8828. DOI: 10.1109/34.777378 (cit. on pp. 132, 224).

[20] Isabelle Bloch and Alessandro Saffiotti. "Why robots should use fuzzy mathematical morphology." In: *Proc. of the 1st Int. ICSC-NAISO Congress on Neuro-Fuzzy Technologies, La Havana, Cuba*. 2002 (cit. on p. 218).

[21] Paul Bloom. *Language and Space*. en. MIT Press, 1999. ISBN: 978-0-262-52266-3 (cit. on p. 68).

[22] F. Bobillo and U. Straccia. "fuzzyDL: An Expressive Fuzzy Description Logic Reasoner." In: *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*. June 2008, pp. 923–930. DOI: 10.1109/FUZZY.2008.4630480 (cit. on p. 114).

[23] Fernando Bobillo and Umberto Straccia. "On Qualified Cardinality Restrictions in Fuzzy Description Logics under Łukasiewicz Semantics." en. In: *Proceedings of the 12th International Conference of Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2008)*. Vol. 8. June 2008, pp. 1008–1015 (cit. on p. 116).

[24] Fernando Bobillo and Umberto Straccia. "The Fuzzy Ontology Reasoner fuzzyDL." In: *Knowledge-Based Systems* 95 (Mar. 2016), pp. 12–34. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2015.11.017 (cit. on pp. 40, 69, 105, 137).

[25] Razvan Gabriel Boboc, Adrian Iulian Dumitru, and Csaba Antonya. "Point-and-command paradigm for interaction with assistive robots." In: *International Journal of Advanced Robotic Systems* 12.6 (2015), p. 75 (cit. on p. 146).

[26] Christopher Brinton. "A Framework for Explanation of Machine Learning Decisions." In: *IJCAI-17 Workshop on Explainable AI (XAI)*. 2017, pp. 14–18 (cit. on p. 147).

[27] Gordon D. A. Brown, Ian Neath, and Nick Chater. "A Temporal Ratio Model of Memory." en. In: *Psychological Review* 114.3 (July 2007), pp. 539–576. ISSN: 0033-295X (cit. on p. 172).

[28] D. Brugali. *Software Engineering for Experimental Robotics*. Heidelberg, Germany: Springer, 2007 (cit. on p. 19).

[29] Lorenz Bühmann, Jens Lehmann, Patrick Westphal, and Simon Bin. "DL-Learner Structured Machine Learning on Semantic Web Data." en. In: *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*. Lyon, France: ACM Press, 2018, pp. 467–471. ISBN: 978-1-4503-5640-4. DOI: 10.1145/3184558.3186235 (cit. on pp. 4, 40, 112, 113, 211).

[30] Luca Buoncompagni, Barbara Bruno, Antonella Giuni, Fulvio Mastrogiovanni, and Renato Zaccaria. "Towards a New Paradigm for Assistive Technology at Home: Research Challenges, Design Issues and Performance Assessment." In: *arXiv:1710.10164 [cs]* (Oct. 2017). arXiv: 1710.10164 [cs] (cit. on p. 166).

[31] E. Cambria and B. White. "Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]." In: *IEEE Computational Intelligence Magazine* 9.2 (May 2014), pp. 48–57. ISSN: 1556-603X. DOI: 10.1109/MCI.2014.2307227 (cit. on p. 113).

[32] Alessio Capitanelli and Fulvio Mastrogiovanni. "An Ontology-Based Hybrid Architecture for Planning and Robust Execution in Tabletop Scenarios." en. In: *Proceedings of the 3rd Italian Workshop on Artificial Intelligence and Robotics A Workshop of the XV International Conference of the Italian Association for Artificial Intelligence (AI*IA 2016) November 28, 2016*. Vol. 1834. Genova, Italy: CEUR-WS, Nov. 2016, p. 5 (cit. on pp. 5, 41, 61, 112).

[33] A. Carfi, C. Motolese, B. Bruno, and F. Mastrogiovanni. "Online Human Gesture Recognition Using Recurrent Neural Networks and Wearable Sensors." In: *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. Aug. 2018, pp. 188–195. DOI: 10.1109/ROMAN.2018.8525769 (cit. on p. 111).

[34] Gunnar Carlsson. "Topology and Data." en. In: *Bulletin of the American Mathematical Society* 46.2 (2009), pp. 255–308. ISSN: 0273-0979, 1088-9485. DOI: 10.1090/S0273-0979-09-01249-X (cit. on p. 68).

[35] J. Chamorro-Martínez, D. Sánchez, J. M. Soto-Hidalgo, and P. M. Martínez-Jiménez. "A Discussion on Fuzzy Cardinality and Quantification. Some Applications in Image Processing." In: *Fuzzy Sets and Systems*. Special Issue on Fuzzy Numbers and Their Applications 257 (Dec. 2014), pp. 85–101. ISSN: 0165-0114. DOI: 10.1016/j.fss.2013.05.009 (cit. on p. 117).

[36] Paul Chance. *Learning and Behavior*. en. Cengage Learning, Mar. 2013. ISBN: 978-1-285-54596-7 (cit. on p. 172).

[37] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 77–85. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.16 (cit. on p. 111).

[38] Weihong Chen, Jiyao An, Renfa Li, Li Fu, Guoqi Xie, Md Zakirul Alam Bhuiyan, and Keqin Li. "A Novel Fuzzy Deep-Learning Approach to Traffic Flow Prediction with Uncertain Spatial–Temporal Data Features." In: *Future Generation Computer Systems* 89 (Dec. 2018), pp. 78–88. ISSN: 0167-739X. DOI: 10.1016/j.future.2018.06.021 (cit. on p. 114).

[39] N. J. Cohen and L. R. Squire. "Preserved Learning and Retention of Pattern-Analyzing Skill in Amnesia: Dissociation of Knowing How and Knowing That." en. In: *Science* 210.4466 (Oct. 1980), pp. 207–210. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.7414331 (cit. on p. 170).

[40] Anthony G. Cohn and Jochen Renz. "Qualitative Spatial Representation and Reasoning." In: *Foundations of Artificial Intelligence*. Ed. by Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. Vol. 3. Handbook of Knowledge Representation. Elsevier, Jan. 2008, pp. 551–596. DOI: 10.1016/S1574-6526(07)03013-1 (cit. on p. 68).

[41] Anthony G. Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. "Qualitative Spatial Representation and Reasoning with the Region Connection Calculus." en. In: *GeoInformatica* 1.3 (Oct. 1997), pp. 275–316. ISSN: 1573-7624. DOI: 10.1023/A:1009712514511 (cit. on p. 68).

[42] Silvia Coradeschi and Alessandro Saffiotti. "An Introduction to the Anchoring Problem." In: *Robotics and Autonomous Systems*. Perceptual Anchoring: Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems 43.2 (May 2003), pp. 85–96. ISSN: 0921-8890. DOI: 10.1016/S0921-8890(03)00021-6 (cit. on pp. 69, 146, 211).

[43] Francisco Cruz, German I Parisi, Johannes Twiefel, and Stefan Wermter. "Multi-modal integration of dynamic audiovisual patterns for an interactive reinforcement learning scenario." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 759–766 (cit. on pp. 147, 217).

[44] Kerstin Dautenhahn. "Socially intelligent robots: dimensions of human–robot interaction." In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 362.1480 (2007), pp. 679–704 (cit. on p. 3).

[45] A. De Luca and S. Termini. "A Definition of a Nonprobabilistic Entropy in the Setting of Fuzzy Sets Theory." In: *Information and Control* 20.4 (May 1972), pp. 301–312. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(72)90199-4 (cit. on p. 117).

[46] Richard Dearden and Chris Burbridge. "Manipulation Planning Using Learned Symbolic State Abstractions." In: *Robotics and Autonomous Systems*. Advances in Autonomous Robotics — Selected extended papers of the joint 2012 TAROS Conference and the FIRA RoboWorld Congress, Bristol, UK 62.3 (Mar. 2014), pp. 355–365. ISSN: 0921-8890. DOI: 10.1016/j.robot.2013.09.015 (cit. on p. 68).

[47] Hannah M. Dee, David C. Hogg, and Anthony G. Cohn. "Scene Modelling and Classification Using Learned Spatial Relations." en. In: *Spatial Information Theory*. Ed. by Kathleen Stewart Hornsby, Christophe Claramunt, Michel Denis, and Gérard Ligozat. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 295–311. ISBN: 978-3-642-03832-7 (cit. on p. 68).

[48] Adrien Delaye and Eric Anquetil. "Learning of fuzzy spatial relations between handwritten patterns." In: *International Journal of Data Mining, Modelling and Management* 2 6.2 (2014), pp. 127–147 (cit. on p. 218).

[49] Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. "Comparison of Reasoners for Large Ontologies in the OWL 2 EL Profile." en. In: *Semantic Web* 2.2 (Jan. 2011), pp. 71–87. ISSN: 1570-0844. DOI: 10.3233/SW-2011-0034 (cit. on p. 169).

[50] Natalia Díaz Rodríguez, Manuel P. Cuéllar, Johan Lilius, and Miguel Delgado Calvo-Flores. "A Fuzzy Ontology for Semantic Modelling and Recognition of Human Behaviour." en. In: *Knowledge-Based Systems* 66 (Aug. 2014), pp. 46–60. ISSN: 09507051. DOI: 10.1016/j.knosys.2014.04.016 (cit. on p. 114).

[51] Yadin Dudai. "The Neurobiology of Consolidations, Or, How Stable Is the Engram?" en. In: *Annual Review of Psychology* 55.1 (Feb. 2004), pp. 51–86. ISSN: 0066-4308, 1545-2085. DOI: 10.1146/annurev.psych.55.090902.142050 (cit. on p. 168).

[52] Markus Eich, Ronny Hartanto, Sebastian Kasperski, Sankaranarayanan Natarajan, and Johannes Wollenberg. "Towards Coordinated Multirobot Missions for Lunar Sample Collection in an Unknown Environment." en. In: *Journal of Field Robotics* 31.1 (2014), pp. 35–74. ISSN: 1556-4967. DOI: 10.1002/rob.21491 (cit. on p. 114).

[53] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. "Multimodal deep learning for robust rgb-d object recognition." In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. La Jolla, California, USA, 2015, pp. 681–687 (cit. on p. 32).

[54] Jeffrey M Ellenbogen, Jessica D Payne, and Robert Stickgold. "The Role of Sleep in Declarative Memory Consolidation: Passive, Permissive, Active or None?" en. In: *Current Opinion in Neurobiology* 16.6 (Dec. 2006), pp. 716–722. ISSN: 09594388. DOI: 10.1016/j.conb.2006.10.006 (cit. on p. 168).

[55] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. "3-D mapping with an RGB-D camera." In: *IEEE Transactions on Robotics* 30.1 (2013), pp. 177–187 (cit. on p. 20).

[56] Arnaud Nguembang Fadja and Fabrizio Riguzzi. "Probabilistic Logic Programming in Action." en. In: *Towards Integrative Machine Learning and Knowledge Extraction*. Ed. by Andreas Holzinger, Randy Goebel, Massimo Ferri, and Vasile Palade. Lecture Notes in Computer Science. Springer International Publishing, 2017, pp. 89–116. ISBN: 978-3-319-69775-8 (cit. on p. 68).

[57] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. "Induction of Concepts in Web Ontologies through Terminological Decision Trees." en. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by José Luis Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 442–457. ISBN: 978-3-642-15880-3 (cit. on p. 113).

[58] Adrian Farrel. "Routing Backus-Naur Form (RBNF): A Syntax Used to Form Encoding Rules in Various Routing Protocol Specifications." In: *RFC 5511* (2009). DOI: DOI10.17487/RFC5511 (cit. on p. 26).

[59] S. Fichtl, A. McManus, W. Mustafa, D. Kraft, N. Krüger, and F. Guerin. "Learning Spatial Relationships from 3D Vision Using Histograms." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 501–508. DOI: 10.1109/ICRA.2014.6906902 (cit. on p. 68).

[60] Richard E. Fikes and Nils J. Nilsson. "Strips: A New Approach to the Application of Theorem Proving to Problem Solving." en. In: *Artificial Intelligence* 2.3-4 (Dec. 1971), pp. 189–208. ISSN: 00043702. DOI: 10.1016/0004-3702(71)90010-5 (cit. on p. 111).

[61] Terrence Fong, Charles Thorpe, and Charles Baur. "Collaboration, dialogue, human-robot interaction." In: *Robotics Research*. Springer, 2003, pp. 255–266 (cit. on p. 147).

[62] Maria Fox, Derek Long, and Daniele Magazzeni. "Explainable Planning." In: *IJCAI-17 Workshop on Explainable AI (XAI)*. 2017, pp. 24–30 (cit. on p. 148).

[63] S. T. Freedman and J. A. Adams. "Filtering Data Based on Human-Inspired Forgetting." eng. In: *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics: a publication of the IEEE Systems, Man, and Cybernetics Society* 41.6 (Dec. 2011), pp. 1544–1555. ISSN: 1083-4419. DOI: 10.1109/TSMCB.2011.2157142 (cit. on p. 173).

[64] C. Galleguillos, A. Rabinovich, and S. Belongie. "Object Categorization Using Co-Occurrence, Location and Appearance." In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. June 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587799 (cit. on p. 68).

[65] Carolina Galleguillos and Serge Belongie. "Context Based Object Categorization: A Critical Survey." In: *Computer Vision and Image Understanding*. Special Issue on Multi-Camera and Multi-Modal Sensor Fusion 114.6 (June 2010), pp. 712–722. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2010.02.004 (cit. on p. 68).

[66] John M. Gardiner, Berthold Gawlik, and Alan Richardson-Klavehn. "Maintenance Rehearsal Affects Knowing, Not Remembering; Elaborative Rehearsal Affects Remembering, Not Knowing." en. In: *Psychonomic Bulletin & Review* 1.1 (Mar. 1994), pp. 107–110. ISSN: 1069-9384, 1531-5320. DOI: 10.3758/BF03200764 (cit. on p. 168).

[67] E. Bruce Goldstein, Daniel Vanhorn, Greg Francis, and Ian Neath. *Cognitive Psychology : Connecting Mind, Research, and Everyday Experience*. English. Third edition. Belmont, CA : Wadsworth/Cengage Learning, 2011. ISBN: 978-0-8400-3349-9 (cit. on p. 168).

[68] Robert L. Greene. "Effects of Maintenance Rehearsal on Human Memory." In: *Psychological Bulletin* 102 (Nov. 1987), pp. 403–413. DOI: 10.1037/0033-2909.102.3.403 (cit. on p. 168).

[69] David Gunning. "Explainable artificial intelligence (xai)." In: *Defense Advanced Research Projects Agency (DARPA)* (2017) (cit. on p. 146).

[70] S. Haddadin, M. Suppa, S. Fuchs, T. Bodenmuller, A. Albu-Schäffer, and G. Hirzinger. "Towards the robotic co-worker." In: *Proceedings of the 2009 International Symposium on Robotics Research (ISRR 2009)*. Lucerne, Switzerland, 2009 (cit. on p. 18).

[71] Thomas C. Henderson, Anshul Joshi, and Wenyi Wang. *The Cognitive Symmetry Engine*. 2013 (cit. on p. 169).

[72] W. C. Ho, K. Dautenhahn, M. Y. Lim, P. A. Vargas, R. Aylett, and S. Enz. "An Initial Memory Model for Virtual and Robot Companions Supporting Migration and Long-Term Interaction." In: *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*. Sept. 2009, pp. 277–284. DOI: 10.1109/ROMAN.2009.5326204 (cit. on p. 170).

[73] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for OWL Ontologies." en. In: *Semantic Web* 2.1 (Jan. 2011), pp. 11–21. ISSN: 1570-0844. DOI: 10.3233/SW-2011-0025 (cit. on pp. 40, 59, 154, 197).

[74] Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." In: (May 2004) (cit. on pp. 45, 70).

[75] A. Horzyk, J. A. Starzyk, and J. Graham. "Integration of Semantic and Episodic Memories." In: *IEEE Transactions on Neural Networks and Learning Systems* 28.12 (Dec. 2017), pp. 3084–3095. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2017.2728203 (cit. on p. 171).

[76] Chien-Ming Huang and Bilge Mutlu. "Learning-based modeling of multimodal behaviors for humanlike robots." In: *Proceedings of the ACM/IEEE international conference on Human-robot interaction*. ACM. 2014, pp. 57–64 (cit. on p. 218).

[77] Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. "An Algorithm Based on Counterfactuals for Concept Learning in the Semantic Web." en. In: *Applied Intelligence* 26.2 (Apr. 2007), pp. 139–159. ISSN: 1573-7497. DOI: 10.1007/s10489-006-0011-5 (cit. on p. 113).

[78] Josue Iglesias and Jens Lehmann. "Towards Integrating Fuzzy Logic Capabilities into an Ontology-Based Inductive Logic Programming Framework." en. In: *11th International Conference on Intelligent Systems Design and Applications*. Cordoba, Spain: IEEE, Nov. 2011, pp. 1323–1328. ISBN: 978-1-4577-1676-8 978-1-4577-1675-1. DOI: 10.1109/ISDA.2011.6121843 (cit. on p. 114).

[79] A. Jain, D. Das, J. K. Gupta, and A. Saxena. "PlanIt: A crowdsourcing approach for learning to plan paths from large scale preference feedback." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 877–884. DOI: 10.1109/ICRA.2015.7139281 (cit. on p. 112).

[80] John Jonides, Richard L. Lewis, Derek Evan Nee, Cindy A. Lustig, Marc G. Berman, and Katherine Sledge Moore. "The Mind and Brain of Short-Term Memory." In: *Annual review of psychology* 59 (2008), pp. 193–224. ISSN: 0066-4308. DOI: 10.1146/annurev.psych.59.103006.093615 (cit. on p. 172).

[81] Raymond W. Gibbs Jr. *Embodiment and Cognitive Science*. en. Cambridge University Press, Dec. 2005. ISBN: 978-1-139-44738-6 (cit. on pp. 3, 66, 109, 166).

[82] Peter H. Kahn, Nathan G. Freier, Takayuki Kanda, Hiroshi Ishiguro, Jolina H. Ruckert, Rachel L. Severson, and Shaun K. Kane. "Design Patterns for Sociality in Human-Robot Interaction." In: *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*. HRI '08. New York, NY, USA: ACM, 2008, pp. 97–104. ISBN: 978-1-60558-017-3. DOI: 10.1145/1349822.1349836 (cit. on pp. 6, 166).

[83] Syed Yusha Kareem, Luca Buoncompagni, and Fulvio Mastrogiovanni. "Arianna+: Scalable Human Activity Recognition by Reasoning with a Network of Ontologies." en. In: *AI*IA 2018 – Advances in Artificial Intelligence*. Ed. by Chiara Ghidini, Bernardo Magnini, Andrea Passerini, and Paolo Traverso. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 83–95. ISBN: 978-3-030-03840-3 (cit. on pp. 4, 43, 69, 105, 171).

[84] A. Kasper, R. Jäkel, and R. Dillmann. "Using Spatial Relations of Objects in Real World Scenes for Scene Structuring and Scene Understanding." In: *2011 15th International Conference on Advanced Robotics (ICAR)*. June 2011, pp. 421–426. DOI: 10.1109/ICAR.2011.6088634 (cit. on p. 68).

[85] Holger Knublauch, Daniel Oberle, Phil Tetlow, and Evan Wallace. *A Semantic Web Primer for Object-Oriented Software Developers*. W3C Working Group Note 9 March 2006. W3C, 2006 (cit. on p. 40).

[86] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement Learning in Robotics: A Survey." en. In: *The International Journal of Robotics Research* 32.11 (Sept. 2013). cites: lenzDeepLearningDetecting2015, pp. 1238–1274. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364913495721 (cit. on p. 168).

[87] Seiji Koide and Hideaki Takeda. "OWL-Full Reasoning from an Object Oriented Perspective." en. In: *The Semantic Web – ASWC 2006*. Ed. by Riichiro Mizoguchi, Zhongzhi Shi, and Fausto Giunchiglia. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 263–277. ISBN: 978-3-540-38331-4 (cit. on p. 41).

[88] Paul Lamere, Philip Kwok, Evandro Gouvea, Bhiksha Raj, Rita Singh, William Walker, Manfred Warmuth, and Peter Wolf. "The CMU SPHINX-4 speech recognition system." In: *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong*. Vol. 1. Citeseer. 2003, pp. 2–5 (cit. on p. 26).

[89] Jean-Baptiste Lamy. "Owlready: Ontology-Oriented Programming in Python with Automatic Classification and High Level Constructs for Biomedical Ontologies." en. In: *Artificial Intelligence in Medicine* 80 (July 2017), pp. 11–28. ISSN: 09333657. DOI: 10.1016/j.artmed.2017.07.002 (cit. on p. 41).

[90] Marián Lekavý and Pavol Návrat. "Expressivity of STRIPS-Like and HTN-Like Planning." en. In: *Agent and Multi-Agent Systems: Technologies and Applications*. Ed. by Ngoc Thanh Nguyen, Adam Grzech, Robert J. Howlett, and Lakhmi C. Jain. Vol. 4496. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 121–130. ISBN: 978-3-540-72829-0 978-3-540-72830-6. DOI: 10.1007/978-3-540-72830-6_13 (cit. on p. 111).

[91] S. Lemaignan, R. Ros, R. Alami, and M. Beetz. "What Are You Talking about? Grounding Dialogue in a Perspective-Aware Robotic Architecture." In: *RO-MAN*. Atlanta, GA, USA: IEEE, July 2011, pp. 107–112. DOI: 10.1109/ROMAN.2011.6005249 (cit. on p. 169).

[92] Séverin Lemaignan, Raquel Ros, E Akin Sisbot, Rachid Alami, and Michael Beetz. "Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction." In: *International Journal of Social Robotics* 4.2 (2012), pp. 181–199 (cit. on p. 148).

[93] Séverin Lemaignan, Mathieu Warnier, E. Akin Sisbot, Aurélie Clodic, and Rachid Alami. "Artificial Cognition for Social Human–Robot Interaction: An Implementation." In: *Artificial Intelligence*. Special Issue on AI and Robotics 247 (June 2017), pp. 45–69. ISSN: 0004-3702. DOI: 10.1016/j.artint.2016.07.002 (cit. on pp. 5, 40, 68).

[94] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep Learning for Detecting Robotic Grasps." en. In: *The International Journal of Robotics Research* 34.4-5 (Apr. 2015), pp. 705–724. ISSN: 0278-3649. DOI: 10.1177/0278364914549607 (cit. on p. 168).

[95] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection." en. In: *The International Journal of Robotics Research* 37.4-5 (Apr. 2018), pp. 421–436. ISSN: 0278-3649. DOI: 10.1177/0278364917710318 (cit. on p. 111).

[96] X. Liang, L. Lee, and E. P. Xing. "Deep Variation-Structured Reinforcement Learning for Visual Relationship and Attribute Detection." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 4408–4417. DOI: 10.1109/CVPR.2017.469 (cit. on pp. 69, 111).

[97] S. Matthew Liao and Anders Sandberg. "The Normativity of Memory Modification." en. In: *Neuroethics* 1.2 (July 2008), pp. 85–99. ISSN: 1874-5504. DOI: 10.1007/s12152-008-9009-5 (cit. on p. 170).

[98] Francesca A. Lisi and Umberto Straccia. "A FOIL-Like Method for Learning under Incompleteness and Vagueness." en. In: *Inductive Logic Programming*. Ed. by Gerson Zaverucha, Vítor Santos Costa, and Aline Paes. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 123–139. ISBN: 978-3-662-44923-3 (cit. on p. 114).

[99] D. Liu, M. Cong, and Y. Du. "Episodic Memory-Based Robotic Planning Under Uncertainty." In: *IEEE Transactions on Industrial Electronics* 64.2 (Feb. 2017), pp. 1762–1772. ISSN: 0278-0046. DOI: 10.1109/TIE.2016.2613507 (cit. on p. 169).

[100] Hongyi Liu, Tongtong Fang, Tianyu Zhou, Yuquan Wang, and Lihui Wang. "Deep Learning-Based Multimodal Control Interface for Human-Robot Collaboration." en. In: *Procedia CIRP* 72 (2018), pp. 3–8. ISSN: 22128271. DOI: 10.1016/j.procir.2018.03.224 (cit. on p. 111).

[101] Lorenzo Lucignano, Francesco Cutugno, Silvia Rossi, and Alberto Finzi. "A dialogue system for multimodal human-robot interaction." In: *Proceedings of the 15th ACM on International conference on multimodal interaction*. ACM. 2013, pp. 197–204 (cit. on p. 217).

[102] Geoffrey B. Maddox, David A. Balota, Jennifer H. Coane, and Janet M. Duchek. "The Role of Forgetting Rate in Producing a Benefit of Expanded over Equal Spaced Retrieval in Young and Older Adults." In: *Psychology and aging* 26.3 (Sept. 2011), pp. 661–670. ISSN: 0882-7974. DOI: 10.1037/a0022942 (cit. on p. 171).

[103] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. "Cloth Grasp Point Detection Based on Multiple-View Geometric Cues with Application to Robotic Towel Folding." In: *2010 IEEE International Conference on Robotics and Automation*. May 2010, pp. 2308–2315. DOI: 10.1109/ROBOT.2010.5509439 (cit. on p. 68).

[104] Stephen Maren. "Long-Term Potentiation in the Amygdala: A Mechanism for Emotional Learning and Memory." en. In: *Trends in Neurosciences* 22.12 (Dec. 1999), pp. 561–567. ISSN: 01662236. DOI: 10.1016/S0166-2236(99)01465-4 (cit. on p. 168).

[105] I. Maurtua, I. Fernandez, J. Kildal, L. Susperregi, A. Tellaeche, and A. Ibarguren. "Enhancing Safe Human-Robot Collaboration through Natural Multimodal Communication." In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2016, pp. 1–8. DOI: 10.1109/ETFA.2016.7733573 (cit. on p. 112).

[106] Mausam and Andrey Kolobov. "Planning with Markov Decision Processes: An AI Perspective." In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6.1 (June 2012), pp. 1–210. ISSN: 1939-4608. DOI: 10.2200/S00426ED1V01Y201206AIM017 (cit. on p. 111).

[107] James L McGaugh and Benno Roozendaal. "Role of Adrenal Stress Hormones in Forming Lasting Memories in the Brain." en. In: *Current Opinion in Neurobiology* 12.2 (Apr. 2002), pp. 205–210. ISSN: 09594388. DOI: 10.1016/S0959-4388(02)00306-9 (cit. on p. 168).

[108] Deborah L. McGuinness and Frank Van Harmelen. "OWL Web Ontology Language Overview." en. In: *W3C Recommendation* 10.10 (2004), p. 22 (cit. on pp. 4, 21, 40, 67, 112).

[109] Douglas L. Medin, ed. *The Psychology of Learning and Motivation: Advances in Research and Theory Volume 31*. English. Academic Press, July 2011. ISBN: 978-0-12-399405-9 (cit. on p. 169).

[110] Vahid Mokhtari, Luis Seabra Lopes, and Armando J. Pinho. "Experience-Based Robot Task Learning and Planning with Goal Inference." en. In: *Twenty-Sixth International Conference on Automated Planning and Scheduling*. Mar. 2016 (cit. on p. 111).

[111] Reinhard Moratz, Thora Tenbrink, John Bateman, and Kerstin Fischer. "Spatial Knowledge Representation for Human-Robot Interaction." en. In: *Spatial Cognition III*. Ed. by Christian Freksa, Wilfried Brauer, Christopher Habel, and Karl F. Wender. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 263–286. ISBN: 978-3-540-45004-7 (cit. on p. 68).

[112] R. G. Morris. "D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949." eng. In: *Brain Research Bulletin* 50.5-6 (1999 Nov-Dec), p. 437. ISSN: 0361-9230 (cit. on p. 169).

[113] Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, and Uli Sattler. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. en. Tech. rep. 65-27. W3C recommendation, 2009, p. 134 (cit. on p. 45).

[114] Andrew Nuxoll and John E. Laird. "A Cognitive Model of Episodic Memory Integrated with a General Cognitive Architecture." In: *ICCM*. 2004 (cit. on p. 171).

[115] M. Oliveira, G. H. Lim, L. S. Lopes, S. H. Kasaei, A. M. Tomé, and A. Chauhan. "A Perceptual Memory System for Grounding Semantic Representations in Intelligent Service Robots." In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 2216–2223. DOI: 10.1109/IROS.2014.6942861 (cit. on pp. 169, 171).

[116] Dan R Olsen and Michael A Goodrich. "Metrics for evaluating human-robot interactions." In: *Proceedings of PERMIS*. Vol. 2003. 2003, p. 4 (cit. on p. 217).

[117] Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013 (cit. on p. 27).

[118] Mike Phillips, Victor Hwang, Sachin Chitta, and Maxim Likhachev. "Learning to Plan for Constrained Manipulation from Demonstrations." en. In: *Autonomous Robots* 40.1 (Jan. 2016), pp. 109–124. ISSN: 1573-7527. DOI: 10.1007/s10514-015-9440-5 (cit. on p. 111).

[119] G. Pointeau, M. Petit, and P. F. Dominey. "Successive Developmental Levels of Autobiographical Memory for Learning Through Social Interaction." In: *IEEE Transactions on Autonomous Mental Development* 6.3 (Sept. 2014), pp. 200–212. ISSN: 1943-0604. DOI: 10.1109/TAMD.2014.2307342 (cit. on p. 171).

[120] Alberto Poncela and Leticia Gallardo-Estrella. "Command-based voice teleoperation of a mobile robot via a human-robot interface." In: *Robotica* 33.1 (2015), pp. 1–18 (cit. on p. 217).

[121] André Potenza, Andrey Kiselev, Amy Loutfi, and Alessandro Saffiotti. "Towards Sliding Autonomy in Mobile Robotic Telepresence: A Position Paper." In: *ECCE 2017-European Conference on Cognitive Ergonomics, 20-22 September 2017, Umeå University, Sweden*. 2017 (cit. on p. 216).

[122] F. Pratama, F. Mastrogiovanni, and N. Y. Chong. "An Integrated Epigenetic Robot Architecture via Context-Influenced Long-Term Memory." In: *4th International Conference on Development and Learning and on Epigenetic Robotics*. Oct. 2014, pp. 68–74. DOI: 10.1109/DEVLRN.2014.6982956 (cit. on p. 66).

[123] F. Pratama, F. Mastrogiovanni, S. Jeong, and N. Y. Chong. "Long-Term Knowledge Acquisition in a Memory-Based Epigenetic Robot Architecture for Verbal Interaction." In: *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. Aug. 2015, pp. 25–30. DOI: 10.1109/ROMAN.2015.7333563 (cit. on p. 66).

[124] Ferdian Pratama, Fulvio Mastrogiovanni, Soon Geul Lee, and Nak Young Chong. "Long-Term Knowledge Acquisition Using Contextual Information in a Memory-Inspired Robot Architecture." In: *Journal of Experimental & Theoretical Artificial Intelligence* 29.2 (Mar. 2017), pp. 313–334. ISSN: 0952-813X. DOI: 10.1080/0952813X.2015.1134679 (cit. on p. 169).

[125] Tony J Prescott, Ben Mitchinson, and Sebastian Conran. "MiRo: An Animal-like Companion Robot with a Biomimetic Brain-based Control System." In: *Proceedings of the Companion of the ACM/IEEE International Conference on Human-Robot Interaction*. ACM. 2017, pp. 50–51 (cit. on pp. 216, 225).

[126] J. R. Quinlan. "Learning Logical Definitions from Relations." en. In: *Machine Learning* 5.3 (Aug. 1990), pp. 239–266. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00117105 (cit. on p. 113).

[127] Jette Randløv and Preben Alstrøm. "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping." en. In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., July 1998, pp. 463–471. ISBN: 1-55860-556-8 (cit. on p. 168).

[128] Daniele Ravi, Charence Wong, Benny Lo, and Guang-Zhong Yang. "A Deep Learning Approach to On-Node Sensor Data Analytics for Mobile or Wearable Devices." en. In: *IEEE Journal of Biomedical and Health Informatics* 21.1 (Jan. 2017), pp. 56–64. ISSN: 2168-2194, 2168-2208. DOI: 10.1109/JBHI.2016.2633287 (cit. on p. 111).

[129] Benjamin Rosman and Subramanian Ramamoorthy. "Learning Spatial Relationships between Objects." en. In: *The International Journal of Robotics Research* 30.11 (Sept. 2011), pp. 1328–1342. ISSN: 0278-3649. DOI: 10.1177/0278364911408155 (cit. on p. 68).

[130] Robert J Ross, Hui Shi, Tillman Vierhuff, Bernd Krieg-Brückner, and John Bateman. "Towards dialogue based shared control of navigating robots." In: *International Conference on Spatial Cognition*. Springer. 2004, pp. 478–499 (cit. on p. 217).

[131] Jonas Rothfuss, Fabio Ferreira, Eren Erdal Aksoy, You Zhou, and Tamim Asfour. "Deep Episodic Memory: Encoding, Recalling, and Predicting Episodic Experiences for Robot Action Execution." en. In: *arXiv:1801.04134 [cs]* (Jan. 2018). arXiv: 1801.04134 [cs] (cit. on p. 168).

[132] R. B. Rusu and S. Cousins. "3D Is Here: Point Cloud Library (PCL)." In: *2011 IEEE International Conference on Robotics and Automation*. May 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567 (cit. on pp. 19, 22, 131, 149).

[133] R. Rusu. *Semantic 3D object maps for everyday robot manipulation*. Heidelberg, Germany: Springer, 2013 (cit. on p. 19).

[134] Ashutosh Saxena, Ashesh Jain, Ozan Sener, Aditya Jami, Dipendra K. Misra, and Hema S. Koppula. "RoboBrain: Large-Scale Knowledge Engine for Robots." en. In: *arXiv:1412.0691 [cs]* (Dec. 2014). arXiv: 1412.0691 [cs] (cit. on pp. 4, 111, 211).

[135] Brian Scassellati. "Theory of mind for a humanoid robot." In: *Autonomous Robots* 12.1 (2002), pp. 13–24 (cit. on p. 3).

[136] Craig Schlenoff, Edson Prestes, Raj Madhavan, Paulo Goncalves, Howard Li, Stephen Balakirsky, Thomas Kramer, and Emilio Miguelanez. "An IEEE Standard Ontology for Robotics and Automation." en. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vilamoura-Algarve, Portugal: IEEE, Oct. 2012, pp. 1337–1342. ISBN: 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. DOI: 10.1109/IROS.2012.6385518 (cit. on p. 40).

[137] R. Schnabel, R. Wahl, and R. Klein. "Efficient RANSAC for point-cloud shape detection." In: *Computer Graphics Forum* 26.2 (2007), pp. 214–226 (cit. on p. 18).

[138] Bechhofer Sean, Van Harmelen Frank, Hendler Jim, Horrocks Ian, McGuinness Deborah L, Patel-Schneider Peter F, and Stein Lynn Andrea. "OWL Web Ontology Language Reference." en. In: *W3C recommendation* 10.2 (2014), p. 80 (cit. on p. 169).

[139] Lanbo She, Shaohua Yang, Yu Cheng, Yunyi Jia, Joyce Chai, and Ning Xi. "Back to the blocks world: Learning new actions through situated human-robot dialogue." In: *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. Springer, 2014, pp. 89–97 (cit. on p. 147).

[140] Raymond Sheh and Isaac Monteath. "Defining Explainable AI for Requirements Analysis." en. In: *KI - Künstliche Intelligenz* 32.4 (Nov. 2018), pp. 261–266. ISSN: 1610-1987. DOI: 10.1007/s13218-018-0559-3 (cit. on p. 112).

[141] Glen Shires and Hans Wennborg. "Web speech api specification." In: *Final Report, W3C* (2012) (cit. on pp. 27, 155).

[142] M. Sigalas, M. Maniadakis, and P. Trahanias. "Episodic Memory Formulation and Its Application in Long-Term HRI." In: *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. Aug. 2017, pp. 599–606. DOI: 10.1109/ROMAN.2017.8172364 (cit. on p. 169).

[143] Evren Sirin and Bijan Parsia. "SPARQL-DL: SPARQL Query for OWL-DL." In: *OWLED*. 2007 (cit. on pp. 45, 70).

[144] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. "Pellet: A Practical OWL-DL Reasoner." In: *Journal of Web Semantics*. Software Engineering and the Semantic Web 5.2 (June 2007), pp. 51–53. ISSN: 1570-8268. DOI: 10.1016/j.websem.2007.03.004 (cit. on pp. 40, 58, 69, 112, 152, 197).

[145] Steven M. Smith. "Environmental Context—Dependent Memory." In: *Memory in Context: Context in Memory*. Oxford, England: John Wiley & Sons, 1988, pp. 13–34. ISBN: 978-0-471-91901-8 (cit. on p. 172).

[146] Cees G.M. Snoek, Marcel Worring, and Arnold W.M. Smeulders. "Early versus late fusion in semantic video analysis." In: *Proceedings of the 13th annual ACM international conference on Multimedia*. ACM. Singapore, 2005, pp. 399–402 (cit. on p. 32).

[147] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. "Parsing Natural Scenes and Natural Language with Recursive Neural Networks." In: (Nov. 2018) (cit. on p. 69).

[148] Larry R. Squire. "Memory Systems of the Brain: A Brief History and Current Perspective." eng. In: *Neurobiology of Learning and Memory* 82.3 (Nov. 2004), pp. 171–177. ISSN: 1074-7427. DOI: 10.1016/j.nlm.2004.06.005 (cit. on p. 170).

[149] P. H. D. Arjuna S. Srimal, M. A. Viraj J. Muthugala, and A. G. Buddhika P. Jayasekara. "Deictic Gesture Enhanced Fuzzy Spatial Relation Grounding in Natural Language." en. In: *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. Naples, Italy: IEEE, July 2017, pp. 1–8. ISBN: 978-1-5090-6034-4. DOI: 10.1109/FUZZ-IEEE.2017.8015637 (cit. on pp. 114, 218).

[150] D. Stachowicz and G. M. Kruijff. "Episodic-Like Memory for Cognitive Robots." In: *IEEE Transactions on Autonomous Mental Development* 4.1 (Mar. 2012), pp. 1–16. ISSN: 1943-0604. DOI: 10.1109/TAMD.2011.2159004 (cit. on p. 169).

[151] Umberto Straccia. "Fuzzy Logic, Annotation Domains and Semantic Web Languages." en. In: *Scalable Uncertainty Management*. Ed. by Salem Benferhat and John Grant. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 2–21. ISBN: 978-3-642-23963-2 (cit. on p. 123).

[152] L. Tai, G. Paolo, and M. Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 31–36. DOI: 10.1109/IROS.2017.8202134 (cit. on p. 111).

[153] Helen Tam, Christopher Jarrold, Alan D. Baddeley, and Maura Sabatos-DeVito. "The Development of Memory Maintenance: Children's Use of Phonological Rehearsal and Attentional Refreshment in Working Memory Tasks." en. In: *Journal of Experimental Child Psychology* 107.3 (Nov. 2010), pp. 306–324. ISSN: 00220965. DOI: 10.1016/j.jecp.2010.05.006 (cit. on p. 168).

[154] Jiacheng Tan, Zhaojie Ju, and Honghai Liu. "Grounding spatial relations in natural language by fuzzy representation for human-robot interaction." In: *Fuzzy Systems (FUZZ-IEEE), IEEE International Conference on*. IEEE. 2014, pp. 1743–1750 (cit. on p. 218).

[155] Moritz Tenorth and Michael Beetz. "KNOWROB – knowledge processing for autonomous personal robots." In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 4261–4266. DOI: 10.1109/IROS.2009.5354602 (cit. on p. 58).

[156] Moritz Tenorth and Michael Beetz. "KnowRob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots." en. In: *The International Journal of Robotics Research* 32.5 (Apr. 2013), pp. 566–590. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364913481635 (cit. on p. 111).

[157] Akshaya Thippur, C. Burbridge, L. Kunze, Marina Alberti, John Folkesson, Patric Jensfelt, and N. Hawes. "A Comparison of Qualitative and Metric Spatial Relation Models for Scene Understanding." eng. In: *29th AAAI Conference on Artificial Intelligence, AAAI 2015 and the 27th Innovative Applications of Artificial Intelligence Conference, IAAI 2015; Austin; United States*. Vol. 2. AI Access Foundation, 2015, pp. 1632–1640 (cit. on p. 68).

[158] Endel Tulving and Tannis Y. Arbuckle. "Input and Output Interference in Short-Term Associative Memory." ENGLISH. In: *Journal of Experimental Psychology* 72.1 (July 1966), pp. 145–150. ISSN: 0022-1015 (cit. on p. 172).

[159] Endel Tulving, Wayne Donaldson, Gordon H. Bower, and United States Office of Naval Research. *Organization of Memory*. en. Academic Press, 1972 (cit. on p. 170).

[160] Benton J. Underwood. "Interference and Forgetting." In: *Psychological Review* 64.1 (Jan. 1957), pp. 49–60. ISSN: 0033-295X (cit. on p. 172).

[161] David Vernon. *Artificial Cognitive Systems: A Primer*. en. MIT Press, Oct. 2014. ISBN: 978-0-262-02838-7 (cit. on pp. 3, 167).

[162] H Wache, T Vögele, U Visser, H Stuckenschmidt, G Schuster, H Neumann, and S Hübner. "Ontology-Based Integration of Information — a Survey of Existing Approaches." In: *IJCAI–01 Workshop: Ontologies and Information Sharing*. Ed. by H Stuckenschmidt. 2001, pp. 108–117 (cit. on p. 40).

[163] Mirko Wächter, Ekaterina Ovchinnikova, Valerij Wittenbeck, Peter Kaiser, Sandor Szedmak, Wail Mustafa, Dirk Kraft, Norbert Krüger, Justus Piater, and Tamim Asfour. "Integrating multi-purpose natural language understanding, robot's memory, and symbolic planning for task execution in humanoid robots." In: *Robotics and Autonomous Systems* 99 (2018), pp. 148–165 (cit. on p. 147).

[164] R. Walecki, O. Rudovic, V. Pavlovic, B. Schuller, and M. Pantic. "Deep Structured Learning for Facial Action Unit Intensity Estimation." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 5709–5718. DOI: 10.1109/CVPR.2017.605 (cit. on p. 69).

[165] M. L. Walters, D. S. Syrdal, K. Dautenhahn, and K. L. Koay I. Boekhorst. "Avoiding the uncanny valley: robot appearance, personality and consistency of behaviours in an attention-seeking home scenario for a robot companion." In: *Autonomous Robots* 24.2 (2008), pp. 159–178 (cit. on p. 18).

[166] T. Wang and Q. Chen. "Object Semantic Map Representation for Indoor Mobile Robots." In: *Proceedings 2011 International Conference on System Science and Engineering*. June 2011, pp. 309–313. DOI: 10.1109/ICSSE.2011.5961919 (cit. on p. 112).

[167] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. "Kintinuous: spatially extended KinectFusion." In: *Proceedings of the 2012 RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia, 2012 (cit. on p. 19).

[168] Rachel Wood, Paul Baxter, and Tony Belpaeme. "A Review of Long-Term Memory in Natural and Synthetic Systems." en. In: *Adaptive Behavior* 20.2 (Apr. 2012), pp. 81–103. ISSN: 1059-7123, 1741-2633. DOI: 10.1177/1059712311421219 (cit. on pp. 166, 167).

[169] F. Wörgötter et al. "Structural Bootstrapping—A Novel, Generative Mechanism for Faster and More Efficient Acquisition of Action-Knowledge." In: *IEEE Transactions on Autonomous Mental Development* 7.2 (June 2015), pp. 140–154. ISSN: 1943-0604. DOI: 10.1109/TAMD.2015.2427233 (cit. on p. 108).

[170] P. Ye, T. Wang, and F. Wang. "A Survey of Cognitive Architectures in the Past 20 Years." In: *IEEE Transactions on Cybernetics* 48.12 (Dec. 2018), pp. 3280–3290. ISSN: 2168-2267. DOI: 10.1109/TCYB.2018.2857704 (cit. on p. 173).

[171] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. "The HTK book." In: *Cambridge university engineering department* 3 (2002), p. 175 (cit. on p. 26).

[172] R. Zese. *Probabilistic Semantic Web: Reasoning and Learning*. en. Vol. 28. Studies on the semantic web. IOS Press, Dec. 2016. ISBN: 978-1-61499-734-4 (cit. on pp. 40, 69, 113).

[173] J. A. Zlotowski, H. Sumioka, S. Nishio, D. F. Glas, C. Bartneck, and H. Ishiguro. "Persistence of the uncanny valley: the influence of repeated interactions and a robot's attitude on its perception." In: *Frontiers in Psychology* 6 (2015), p. 883 (cit. on p. 18).